MIXED CENTRALIZED/DECENTRALIZED COORDINATION PROTOCOLS FOR MULTI-AGENT SYSTEMS

A Dissertation
Presented to
The Academic Faculty

By

Matthew Thomas Hale

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

May 2017

MIXED CENTRALIZED/DECENTRALIZED COORDINATION PROTOCOLS FOR MULTI-AGENT SYSTEMS

Approved by:

Dr. Magnus Egerstedt, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Yorai Wardi, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Aaron Ames
Department of Mechanical and Civil Engineering
California Institute of Technology

Dr. Mark Davenport
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Eric Feron
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Jeff Shamma
Computer, Electrical and Mathematical Sciences & Engineering Division
King Abdullah University of Science and Technology

Date Approved: March 29, 2017

Research is what I'm doing when I don't know what I'm doing

Wernher von Braun

To everyone who helped along the way

ACKNOWLEDGMENTS

The first thing I want to say is that grad school has been a blast. The combination of work I got to do and people I got to meet has let me enjoy essentially all of the last five years. For that I feel very fortunate, and my goal here is to thank everyone who made things go so smoothly.

The very first people I'd like to thank are my family. My mom, dad, and Julie were always encouraging of what I was doing, even though I don't think I ever did a good job of explaining exactly what that was or why it was so important to me. Over time, they seemed to have unbounded patience and enthusiasm, even during times when I was severely limited in both, and for that I am extremely grateful. They always emphasized the importance of getting an education and that played a vital role in getting me to this point. Without their support, I can truthfully say that I would not have made it to graduation. The same goes for my Aunt Mary Ann and Uncle Glenn. They were certain that I was doing something worthwhile and likewise always encouraged me in whatever I chose to do. Knowing their door was always open was very comforting, and I made good use of that whenever I could. Visiting them also gave me a chance to see my grandparents, who live nearby. Whenever I couldn't see them, I did my best to call regularly, and my conversations with them were constantly encouraging and really, really funny, so that a number of frustrated, unmotivated days were pretty easily turned around by talking with them. Their unwavering support has meant the world to me. I also talked with my brother Mike on the phone fairly regularly, and the act of deliberately *not* talking about controls or robotics with him was always cathartic.

I also want to thank my lab mates and fellow grad students, both past and present. They're too numerous to name here, but I want to single out by name Zak Costello, Yancy Diaz-Mercado, and Thiagarajan Ramachandran for the many, many useful conversations we've had. Talking with them was always interesting and illuminating, and their insights helped push me to make my work better. With respect to technical work, I can't imagine a better group of people to be around to get ideas from, bounce ideas off of, and to work hard with. My time in the GRITS Lab has included both the most work I've ever done and the most fun I've ever had, and I will sincerely miss coming

into TSRB every day and spending my days here.

Outside of Georgia Tech, I was fortunate enough to have two highly successful collaborations. As my work moved toward multi-agent optimization, I had the chance to travel to UIUC and work with Angelia Nedić. I quickly learned a lot from her, especially about areas of the literature that were previously unknown to me, and this collaboration led to some of the most exciting work I've had the chance to do. During that same trip, I gave a special DCL Seminar at the Coordinated Science Lab, and that led to working on differential privacy with Geir Dullerud and his student Yu Wang. Together, we've produced results that I am very proud of, and I hope that both collaborations continue into the future.

During my time in Atlanta, I've also had the chance to be friends with some amazing people. Zak Costello introduced me to rock climbing and the people who do it, and that it turn let me meet Luke, Claire, Blaine, Mark, and Cheri, as well as the whole Alex, Alyna, Kristen, and Eugene gang from the Marcus Center. I had a lot of fun living here as a result of being part of these circles. I also want to say thanks to the Costello family for always making me feel welcome; at this point, I've invited myself over to (or else just shown up unannounced at) the residence of basically every member of the Costello family, and they've always been happy to have me. That has made my time in Atlanta very enjoyable, and I'm extremely grateful that they were always willing to have me around.

Through a very long sequence of friend-to-friend introductions, I was also introduced to my girlfriend Brianna Petrone. When we first met, she asked how my research was going, and I was immediately excited because (i) I was very flattered and (ii) my life did not consist of much else at that point. As we spent more and more time together, I got better at balancing working with not working, though Brianna was always very patient and understanding when I had to work (as fate would have it, such occasions were common). However, throughout that time Brianna really did keep me sane by being the best climbing partner, hiking partner, and cooking instructor (and lots of other things). It has become sort of a cliché to exaggerate and say "I wouldn't have made it" without someone, but I have the luxury to unhyperbolically say that I really wouldn't have made it

to this point without Brianna. She's the best.

Finally, I'd like to thank my two advisors, Professor Magnus Egerstedt and Professor Yorai Wardi. I know that there is an abundance of grad school horror stories in the world, but I managed to enjoy basically all of my time at Georgia Tech, and this is due in large part to their efforts. Yorai has taught me virtually everything I know about functional analysis, optimal control, and hybrid systems, but, beyond that, our conversations over time have drifted into any subject that we find interesting (and, as it turns out, there are many such subjects). I've learned from Yorai that the world is interesting and control theory is one interesting part of it, though pursuing one interest doesn't have to come at the exclusion of any others, and that is a lesson that I will always carry with me going forward. From Magnus I've learned an incredible amount about graph theory, multi-agent systems, what it takes to be a researcher, and why academia is obviously the right career path for me. Our technical conversations were of course what kept me on track, but perhaps more valuable was talking about wise career choices, how to present research, how to plan a research program, and *why* research is done the way it is. This instilled in me the idea that a researcher doesn't just do research, but, instead, cultivates a research program, and that that involves much more than putting pen to paper. For the last five years, I always felt as though I was working on what I wanted to work on, though somehow Magnus and Yorai made sure that I only wanted to work on things that were actually worthwhile. As I start advising students of my own, I will do my best to emulate them.

I can't claim to have predicted that grad school would have worked out so well, but I'm pleased to say that, here at the end, my grad school experience was optimal, and that I was fortunate enough to work for the right people in the right place at exactly the right time for me. And, given a control theorist's obsession with initial conditions, I am beyond excited to have this ending point of grad school serve as my start for the next big adventure.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The objective of this thesis is to coordinate multi-agent systems in a way which mixes centralized and decentralized architectures and algorithms. A conventional view of networked control theory makes a sharp distinction between centralized coordination, in which a single decision-maker processes all information and makes all decisions, and decentralized coordination, in which each agent has limited information and makes an individual decision based on that information. Centralization offers complete information about a system, though it can often be too slow, in that gathering and processing all information takes more time than is permitted to make a decision, or else no single decision-maker may be capable of enacting all decisions that must be made. On the other hand, decentralization is often faster and scalable with respect to network size, though the limited information present in a decentralized configuration can make coordination tasks difficult. This work blends these two paradigms in order to retain the benefits of each. At a high level, the goal of this work is to have a network of agents rapidly executing a decentralized behavior while a central aggregator slowly gathers all information, processes it, and broadcasts the results. Intuitively, the addition of centralized information into a decentralized network adds information that would otherwise be absent and, even though this information may be slower than the rest of the network, this addition should be useful simply because more information is available in the system. Throughout, the central aggregator in a network is referred to as a cloud computer because cloud computing provides a flexible means to incorporate some centralization into any network. The role of the cloud is to provide centralized information that is useful in coordinating agents.

To illustrate this point, we explore two classes of problems that naturally arise in multi-agent networks. The first is the problem of asynchronous coordination. In networks which have many mutually interfering communications, in which agents are spread far apart, or in which communications are difficult for any reason, information sharing often happens asynchronously. Simultaneously, agents may generate new information at different times, leading to mismatches in when new information becomes available. Under these conditions, agents are not guaranteed to have the same

information available to them at any point in time. What results is a need to coordinate agents even though they may never agree on the state of the network and even though we cannot ever be sure of what information an agent has onboard. Part I is dedicated to solving such problems in a mixed centralized/decentralized way. Chapter 2 provides an asynchronous coordination algorithm and provides convergence theorems that quantify its performance in a generalized setting. Chapter 3 builds upon this work and explores how often the necessary communications occur in a multi-agent network whose communications are modeled by random graphs. Chapter 4 then implements this work on robots, demonstrating its applicability and success in practice despite asynchronous information sharing.

The second problem domain studied is that of private coordination. Some network applications rely on sensitive user data to function, requiring that agents share sensitive information in the course of completing some task, though users may want guarantees that sensitive information will not be revealed to others. The need for privacy in the context of multi-agent coordination gives rise to an inherent tension: coordinating a network of agents inclines them to share all information they have, while preserving user privacy inclines agents toward sharing no information at all. In this setting, "privacy" refers to protecting users' sensitive data, and incorporating privacy requirements into multi-agent coordination requires protecting the exact value of users' data while still making it useful for network coordination tasks. The goal of Part II is to balance the two objectives of privacy and coordination and protect user data as teams of agents are coordinated. To that end, Chapter 5 provides a coordination algorithm for keeping agents' state trajectories private over time, thereby keeping an agent's activities private. Chapter 6 extends this idea and uses a form of privacy to enforce honest information sharing by agents. Then, Chapter 7 presents a coordination framework in which each agent's objective function is kept private as the agents work together. A robotic implementation of private coordination is given in Chapter 8, demonstrating the utility of this work in a practical setting.

Part III then provides conclusions based on this work and gives future directions for research in Chapter 9.

CHAPTER 1

INTRODUCTION

The objective of the research in this thesis is to design and analyze algorithms for coordinating multi-agent systems using a mix of centralized and decentralized network elements. One contribution of this work is the mixed centralized/decentralized network architecture itself, and the blending of the traditionally separate paradigms of centralized and decentralized control. This notion is realized as a networked system with a central aggregator introduced into it. This mixed architecture in turn enables the other main contributions of this work: mixed centralized/decentralized coordination algorithms. We focus on solving two coordination problems that arise naturally in networked systems, namely coordination under asynchronous information sharing and coordination with user privacy requirements. The fundamental premise of this work is that adding a central aggregator into a network should be useful because it is adding information to the system that would otherwise be absent, and, even if this centralized aggregator is slower than the rest of the network, that additional information should be useful. The utility of the central aggregator is displayed in successfully solving asynchronous and private problems despite the additional challenges imposed by asynchrony and privacy. Coordination tasks in this work are encoded as multi-agent optimization problems and then solved using mixed centralized/decentralized optimization algorithms that are inspired by (but distinct from) certain classical techniques in mathematical programming. Below, we describe the motivation for this work and elaborate on its mixed centralized/decentralized approach.

1.1 Multi-Agent Systems

Multi-agent systems can be broadly understood as systems in which there are multiple decision-makers collaborating on some given task. Such systems arise naturally in many disparate fields,

including sensor networks [1]–[4], robotics [5], [6], smart power grids [7], [8], communications [9]–[11], cyber-physical systems [12], and many others. In a network of many agents, a single agent may know the state of some other agents in the network, though it will typically not know the states of all other agents. Consequently, coordinating multi-agent systems necessarily requires interaction laws that enable coordinated decision making under limited information. The study of multi-agent systems represents a distinct branch of systems theory and decision theory precisely because of the limitations on information that typically arise in such systems. Although it can be challenging to coordinate systems under such a limited information setup, crafting coordination algorithms that function under these conditions often leads to algorithms that are scalable, in the sense that an agent does roughly the same amount of computing and communicating regardless of a network's size. Motivated by the broad applicability and flexibility of the algorithms developed, this thesis considers a general model of multi-agent systems and solves coordination problems in a general framework that applies equally well across different application domains.

1.2 Centralized Versus Decentralized and the Cloud

At a high level, a decentralized algorithm is any algorithm in which there is not a single decision-maker processing information and making decisions in a system. Decentralized approaches stand in contrast to centralized approaches, in which a single decision-maker does indeed gather and process all information, and make all decisions. A conventional view of network coordination separates control systems into those which are centralized and those which are decentralized, and these two approaches are often regarded in the literature as being distinct, with an algorithm being classified as strictly belonging to one or the other.

The study of decentralized and distributed problems and algorithms was initiated in part because there are some settings in which no centralized decision-maker has all information that is needed to make all required decisions or, if such information is available, the central decision-maker is unable to enact all decisions that must be made [13]. In other cases, a central decision-maker simply may not be fast or reactive enough to enact all decisions in a timely fashion, e.g.,

for a system whose agents are spread across large distances and in which large amounts of data must be gathered and processed to make a decision. Nonetheless, one may wonder whether some centralized information would be useful in a multi-agent system, even if that centralized information is slower than the rest of the network. This thesis answers this question in the affirmative by showing that occasional injections of centralized information into a decentralized network are indeed useful, even when these injections happen slowly relative to other information exchanges in the network, and even when they rely on old information.

The motivation for mixing centralized and decentralized information is two-fold. First, in some cases there is some amount of centralized information naturally present in a system, such as in some robotic platforms [14], [15]. Second, advances in cloud computing technology make it simple to add a centralized component to a network that does not already have one. Indeed, cloud computing makes it possible to communicate with many agents, gather information from them, process it, and broadcast the results, all remotely and without significant changes to a network. Therefore, the capability to have centralized information is already present in many systems, and making use of it becomes of matter of embracing an architecture that is largely already present. This thesis explores this idea through examining asynchronous coordination and private coordination. Throughout, the centralized aggregator in a system is called "the cloud;" the centralized component of a network need not always be a cloud computer, though the cloud provides a useful abstraction for such centralized components, and we use the term broadly to encompass anything which serves this role.

1.3 Privacy

In recent years, privacy has become a topic of mainstream discussion reaching beyond the research community. Simultaneously, there has been a rapid proliferation of applications dependent upon user data, and strong safeguards are needed in order to protect sensitive data of individual users. For example, the smart power grid is a rapidly emerging technology that relies on sharing granular power usage information among residential homes for network management. Recent studies from

the United States Department of Energy [16] and the European Data Protection Supervisor [17] have demonstrated that sharing such information can reveal when a homeowner is home or away, awake or asleep, how many people are in a home, and sometimes what they are doing. Accordingly, one would like to include user privacy requirements in the act of multi-agent coordination itself, simultaneously satisfying the needs for joint coordination and individual privacy. There is an inherent tension between privacy and coordination in the sense that the desire for privacy inclines agents to share as little information as possible, while the desire to jointly coordinate many agents inclines them to share as much information as possible. Thus one must find some balance between these two objectives to satisfy them both, and Part II explores this balance in several coordination problems with privacy requirements.

1.4 Why Optimization?

The choice of how to model a multi-agent coordination task is itself an important issue, and in this thesis such tasks are represented using optimization problems. There are several factors motivating this choice. The first is that the problems studied here often have a natural mixture of centralized and decentralized elements contained in them, as when each agent has a local objective and the network itself has an associated global objective. The problems considered here also often have constraints on the agents' states, and the setting of optimization naturally accommodates such constraints in a variety of ways. Another reason for using the setting of optimization is that the algorithms developed here should be readily implementable, and, in particular, they should be well-suited to ordinary computing systems. Iterative optimization algorithms naturally give rise to discrete-time algorithms, allowing for direct implementation; this is in contrast to designing a continuous-time algorithm and then needing to verify that its discretization provides the same theoretical guarantees as the continuous time algorithm does. As a result, we proceed in the setting of optimization.

1.5 Organization of the Thesis

This thesis is divided into three parts. Part I studies asynchronous coordination problems and is comprised by Chapters 2, 3, and 4. Part II studies private coordination and is comprised by Chapters 5, 6, 7, and 8. Finally, Part III is comprised by Chapter 9 and contains concluding remarks as well as directions for future research. While some chapters consider similar problems, any required assumptions are restated both to make each chapter self-contained and because some subtle but significant details change across problem formulations.

Part I

Asynchronous Coordination

CHAPTER 2

ASYNCHRONOUS MULTI-AGENT PRIMAL-DUAL OPTIMIZATION

Networked coordination and optimization have been applied across a broad range of application domains, such as sensor networks [1]–[4], robotics [6], smart power grids [7], [8], and communications [9]–[11]. A common feature of some applications is the (sometimes implicit) assumption that communications and computations occur in a synchronous fashion. More precisely, though no agent may have access to all information in a network, the information it does have access to is assumed to be up-to-date, and/or computations onboard the agents are assumed to occur concurrently.

One can envision several reasons why these synchrony assumptions may fail. Communications may interfere with each other, slowing data transmissions, or else they may occur serially over a shared channel, resulting in delays when many messages must be sent. In other cases it may simply be undesirable to stay in constant communication in a network due to the energy required to do so, e.g., within a team of battery-powered robots. Apart from communication delays, it may be the case that some agents produce new data, such as a new state value, faster than other agents, leading to mismatches in update rates and, thus, mismatches in when information becomes available. Regardless of their cause, the resulting delays are often unpredictable in duration, and the timeliness of any piece of information in a network with such delays typically cannot be guaranteed. While one could simply have agents pause their computations while synchronizing information across a network, it has been shown that asynchronous algorithms can outperform their synchronous counterparts which pause to synchronize information [18, Section 6.3.5][19, Section 3.3]. Accordingly, this chapter focuses on asynchronous algorithms for multi-agent optimization.

In particular, this chapter considers multi-agent convex optimization problems that need not be separable, and its structural novelty comes from the introduction of a centralized cloud computer

and its associated communication model. The cloud's role is to aggregate centralized information and perform centralized computations for the agents in the network, and the motivation for including a cloud computer comes from its ability to communicate with many devices and its ability to provide ample processing power remotely. However, the cloud's operations take time to perform specifically because they are centralized, and although the cloud adds centralized information to a network, the price one has to pay for this centralized information is that it is generated slowly. As such, the proposed algorithmic model has to take this slowness into account.

In this chapter we consider problems in which each agent has a local cost and local set constraint, and in which the network itself is associated with a non-separable coupling cost. The agents are moreover subject to non-separable ensemble-level inequality constraints[1] that could, for example, correspond to shared resources. To solve these problems, we consider a primal-dual approach that allows the agents' behavior to be totally asynchronous [18, Chapter 6] (cf. partially asynchronous [18, Chapter 7]), both when communicating among themselves and when transmitting to the cloud. However, we do require that the cloud's transmissions to the agents always keep the dual variable's value synchronized among the agents. This synchrony is verified to be necessary in Section 2.5, where a counterexample shows that allowing the agents to disagree upon the value of the system's dual variable can preclude convergence altogether. The dual variable's value is the lone point of synchrony in the presented algorithm, and all other aspects of the system are designed to strive toward operating as asynchronously as possible in a general optimization setting.

To produce such an algorithm, we apply a Tikhonov regularization to the Lagrangian associated with the problem of interest. This regularization causes the algorithm to only approximately solve optimization problems, and error bounds are provided in terms of the regularization parameters, along with a choice rule for selecting these parameters to enforce any desired error bound. The regularization we use induces a tradeoff between speed and accuracy in the optimization process, and it is shown that requiring a less accurate solution allows the algorithm to converge faster and vice versa.

[1]The work here can include equality constraints without any further changes, though we focus only on inequality constraints for notational simplicity.

We also make use of an existing framework for asynchronous optimization [18, Sections 6.1-6.2][20], which accommodates general unconstrained or set-constrained problems. This framework hinges upon the ability to construct a sequence of sets satisfying certain properties which admit a Lyapunov-like convergence result, and we show that our regularization guarantees the ability to construct this sequence of sets as long as the problem satisfies mild assumptions. We also provide novel convergence rate estimates in both the primal and dual spaces that explicitly account for the delays in the system. The contribution of this work thus consists of an asynchronous primal-dual optimization algorithm together with its convergence rates.

There exists a large corpus of work on multi-agent optimization that is related to the work here. In [18] a range of results are gathered on asynchronous multi-agent optimization (for problems without functional constraints or with linear equality constraints) in Chapters 6 and 7. Earlier work on asynchronous algorithms can be traced back to [21] and [22], which consider fixed points of certain classes of operators. Long-standing optimization algorithms known as the Jacobi and Gauss-Seidel methods are also covered in [18] for linear problems in Section 2.4. Linear consensus type problems are studied in [23], including cases in which identical time delays are associated with the communication channels. The framework in [18, Sections 6.1-6.2] is the most general, and we therefore use it as our starting point for optimization in the primal space.

A key difference between our work and earlier work is that we asynchronously solve general constrained convex optimization problems which, in general, need not satisfy the conditions in [18], [21], [22]. The work in [24] also solves constrained optimization problems asynchronously, though it requires bounded communication delays between agents and has each agent updating both a full primal vector and a full dual vector. In the current chapter, communication delays do not have a uniform bound, and each agent updates only its own state as would be the case, e.g., in a team of robots.

Two other relevant and well-known algorithms of current interest are gossip algorithms and the alternating direction method of multipliers (ADMM). Here we do not consider gossip-type algorithms since they either require synchronous communications among the agents or, in the

asynchronous case, allow only one communication channel to be active at a time [25], and our aim is to support communication models that are as general as possible by allowing any number of links to be active at a time.

In contrast to this, ADMM essentially imposes a Gauss-Seidel structure among the primal updates made by the agents [26]. Related work in [27] presents an asynchronous variant of ADMM, though it requires bounded delays and updates of all primal and dual variables onboard each agent, neither of which are required here. The algorithm we present can be viewed as a method related to ADMM that allows all agent behaviors to be essentially arbitrary in their timing. This provides a great degree of flexibility in the agents' primal updates by not requiring any particular ensemble update rule or bounded delays, or requiring an agent to update all variables in the system.

The remainder of the chapter is organized as follows. Section 2.1 describes the optimization problem to be solved and the regularization used. Then Section 2.2 gives a rule for choosing regularization parameters to limit errors in the system. Next, Section 2.3 provides the asynchronous algorithm that is the main focus of the chapter. Then, Section 2.4 proves convergence of the asynchronous algorithm and provides convergence rates for it. We show in Section 2.5 that synchrony in the dual variable is indeed a necessary condition for convergence. Next, Section 2.6 presents simulation results for the asynchronous algorithm, and finally Section 2.7 provides some concluding remarks for this chapter.

2.1 Multi-Agent Optimization

This section gives a description of the problems under consideration and establishes key notation. To that end, the symbol $\|\cdot\|$ without a subscript always denotes the Euclidean norm. We use the notation x_{-i} to denote a vector $x \in \mathbb{R}^n$ with its i^{th} component removed, i.e.,

$$x_{-i} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n).$$

We also define the index set $[P] := \{1, \ldots, P\}$ for all $P \in \mathbb{N}$, and we will use the term "ensemble" to refer to aspects of the problem that involve all agents.

2.1.1 Problem Statement

This chapter solves convex optimization problems over networks comprised by N agents. The agents are indexed over $i \in [N]$, and agent i has an associated decision variable, $x_i \in \mathbb{R}^{n_i}$, with $n_i \in \mathbb{N}$, and we allow for $n_i \neq n_j$ when $i \neq j$. Each agent has to satisfy a local set constraint, expressed by requiring $x_i \in X_i \subseteq \mathbb{R}^{n_i}$, where we assume the following about each X_i.

Assumption 2.1 *For all $i \in [N]$, the set X_i is non-empty, compact, and convex.* $\lozenge$

Note that Assumption 2.1 allows for box constraints, which are common in multi-agent optimization. We will also refer to the ensemble decision variable of the network, defined as

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \in X := X_1 \times \cdots \times X_N \subseteq \mathbb{R}^n,$$

where $n = \sum_{i \in [N]} n_i$. Assumption 2.1 guarantees that X is also non-empty, compact, and convex.

Agent i seeks to minimize a local objective function $f_i : X_i \to \mathbb{R}$ which depends only upon x_i. Together, the agents also seek to minimize a coupling cost $c : \mathbb{R}^n \to \mathbb{R}$ which depends upon all states and can be non-separable. We impose the following assumption on c and each f_i.

Assumption 2.2 *For all $i \in [N]$, the function f_i is convex and C^2 (twice continuously differentiable) in x_i. The function c is convex and C^2 in x.* $\lozenge$

Gathering these costs gives

$$f(x) = c(x) + \sum_{i \in [N]} f_i(x_i),$$

and when Assumptions 2.1 and 2.2 hold, f has a well-defined minimum value over X.

We consider problems with ensemble-level inequality constraints, namely we require that the inequality

$$g(x) := \begin{pmatrix} g_1(x) \\ \vdots \\ g_m(x) \end{pmatrix} \leq 0$$

hold component-wise, under the following assumption.

Assumption 2.3 *For all $j \in [m]$, the function $g_j : \mathbb{R}^n \to \mathbb{R}$ is convex and C^2 in x.* $\lozenge$

In particular, g_j does not need to be separable for any $j \in [m]$. At the ensemble level, we now have a convex optimization problem, stated below.

Problem 2.1

$$\begin{aligned} &\text{minimize } f(x) \\ &\text{subject to } g(x) \leq 0 \\ &\qquad x \in X. \end{aligned}$$

$\blacklozenge$

Section 2.3 will specify an architecture that provides a mixture of distributed information sharing among agents and centralized information from a cloud computer. As a result, the solution to Problem 2.1 will involve a distributed primal-dual algorithm since such an algorithm is implementable in a natural way on the cloud-based architecture. Towards enabling this algorithm, we enforce Slater's condition [28, Assumption 6.4.2] which enables us to find a compact set that contains the optimal dual point in Problem 2.1.

Assumption 2.4 *(*Slater's condition*) There exists a point $\bar{x} \in X$ such that $g(\bar{x}) < 0$.* $\lozenge$

2.1.2 An Ensemble Variational Inequality Formulation

Under Assumptions 2.1-2.4 we define an ensemble variational inequality in terms of Problem 2.1's Lagrangian. The Lagrangian associated with Problem 2.1 is defined as

$$L(x, \mu) = f(x) + \mu^T g(x),$$

where $\mu \in \mathbb{R}^m_+$ and $\mathbb{R}^m_+$ denotes the non-negative orthant of $\mathbb{R}^m$. By definition, $L(\cdot, \mu)$ is convex for all $\mu \in \mathbb{R}^m_+$ and $L(x, \cdot)$ is concave for all $x \in X$. These properties and the differentiability assumptions placed upon f and g together imply that $\nabla_x L(\cdot, \mu) := \frac{\partial L}{\partial x}(\cdot, \mu)$ and $-\nabla_\mu L(x, \cdot) := -\frac{\partial L}{\partial \mu}(x, \cdot)$ are monotone operators on their respective domains. It is known that Assumptions 2.1-2.4 imply that a point $(\hat{x}, \hat{\mu}) \in X \times \mathbb{R}^m_+$ is a solution to Problem 2.1 if and only if it is a saddle point of L [29], i.e., it maximizes L over μ and minimizes L over x so that it satisfies the inequalities

$$L(\hat{x}, \mu) \leq L(\hat{x}, \hat{\mu}) \leq L(x, \hat{\mu}) \tag{2.1}$$

for all $x \in X$ and $\mu \in \mathbb{R}^m_+$. From Assumptions 2.1-2.4 it is guaranteed that a saddle point $(\hat{x}, \hat{\mu})$ exists [30, Corollary 2.2.10].

Defining the symbol $\hat{z}$ to denote a saddle point via $\hat{z} = (\hat{x}, \hat{\mu})$ and using $z = (x, \mu)$ to denote an arbitrary point in $X \times \mathbb{R}^m_+$, we define the composite gradient operator

$$\Lambda(z) := \Lambda(x, \mu) = \begin{pmatrix} \nabla_x L(x, \mu) \\ -\nabla_\mu L(x, \mu) \end{pmatrix}.$$

Then the saddle point condition in Equation (2.1) can be restated as the following ensemble variational inequality [30, Page 21].

Problem 2.2 *Find a point $\hat{z} \in X \times \mathbb{R}^m_+$ such that $(z - \hat{z})^T \Lambda(\hat{z}) \geq 0$ for all $z \in X \times \mathbb{R}^m_+$.* ♦

2.1.3 Tikhonov Regularization

Instead of solving Problem 2.2 as stated, we regularize the problem in order to make it more readily solved asynchronously and to enable us to analyze the convergence rate of the forthcoming asynchronous algorithm. The remainder of this chapter will make extensive use of this regularization. First we define η-strong convexity for a differentiable function.

Definition 2.1 *A differentiable function f is said to be η-strongly convex if*

$$\big(\nabla f(v_1) - \nabla f(v_2)\big)^T (v_1 - v_2) \geq \eta \|v_1 - v_2\|^2$$

for all v_1 and v_2 in the domain of f. $\Diamond$

We now regularize the Lagrangian using constants $\alpha > 0$ and $\beta > 0$ to get

$$L_{\alpha,\beta}(x, \mu) = f(x) + \frac{\alpha}{2}\|x\|^2 + \mu^T g(x) - \frac{\beta}{2}\|\mu\|^2,$$

where we see that $L_{\alpha,\beta}(\cdot, \mu)$ is α-strongly convex and $L_{\alpha,\beta}(x, \cdot)$ is β-strongly concave (which is equivalent to $-L_{\alpha,\beta}(x, \cdot)$ being β-strongly convex). Accordingly, $\nabla_x L_{\alpha,\beta}(\cdot, \mu)$ and $-\nabla_\mu L_{\alpha,\beta}(x, \cdot)$ are strongly monotone operators over their domains. We also define $\kappa = (\alpha, \beta)$ and replace the subscripts α and β with the single subscript κ for brevity when we are not using specific values of α and β. We now have the regularized composite gradient operator,

$$\Lambda_\kappa(z) := \Lambda_\kappa(x, \mu) = \begin{pmatrix} \nabla_x L_\kappa(x, \mu) \\ -\nabla_\mu L_\kappa(x, \mu) \end{pmatrix}.$$

The strong monotonicity of $\nabla_x L_\kappa(\cdot, \mu)$ and $-\nabla_\mu L_\kappa(x, \cdot)$ together imply that Λ_κ itself is strongly monotone, and Assumptions 2.1-2.4 imply that L_κ has a unique saddle point, $\hat{z}_\kappa$ [30, Theorem 2.3.3]. We now focus on solving the following regularized ensemble variational inequality.

Problem 2.3 *Find the point $\hat{z}_\kappa := (\hat{x}_\kappa, \hat{\mu}_\kappa) \in X \times \mathbb{R}^m_+$ such that $(z - \hat{z}_\kappa)^T \Lambda_\kappa(\hat{z}_\kappa) \geq 0$ for all $z \in X \times \mathbb{R}^m_+$.* $\blacklozenge$

As a result of the regularization of Λ to define Λ_κ, the point $\hat{z}_\kappa$ will not equal $\hat{z}$. In particular, for a solution $\hat{z} = (\hat{x}, \hat{\mu})$ to Problem 2.2 and the solution $\hat{z}_\kappa = (\hat{x}_\kappa, \hat{\mu}_\kappa)$ to Problem 2.3, we will have $\hat{x} \neq \hat{x}_\kappa$ and $\hat{\mu} \neq \hat{\mu}_\kappa$. Thus the regularization done with α and β affords us a greater ability to find saddle points asynchronously and, as will be shown, the ability to estimate convergence rates towards a solution, but does so at the expense of accuracy by changing the solution itself. While solving Problem 2.3 does not result in a solution to Problem 2.2, the continuity of L_κ over $X \times \mathbb{R}^m_+$ suggests that using small values of α and β should lead to small differences between $\hat{z}$ and $\hat{z}_\kappa$ so that the level of error introduced by regularizing is acceptable in many settings. Along these lines, we provide a choice rule for α and β in Section 2.2 that enforces any desired error bound for certain errors due to regularization.

There is a well-established literature regarding projection-based methods for solving variational inequalities like that in Problem 2.3, e.g., [30, Chapter 12.1]. We seek to use projection methods because they naturally fit with the mixed centralized/decentralized architecture to be covered in Section 2.3, though it is required that Λ_κ be Lipschitz to make use of such methods. Currently, Λ_κ cannot be shown to be Lipschitz because its domain, $X \times \mathbb{R}^m_+$, is unbounded. To rectify this situation, we now determine a non-empty, compact, convex set $M \subseteq \mathbb{R}^m_+$ which contains $\hat{\mu}_\kappa$, allowing us to solve Problem 2.3 over a compact domain. Below, we use the unconstrained minimum value of f over X, $f^* := \min_{x \in X} f(x)$, which is well-defined under Assumptions 2.1 and 2.2. We have the following result based upon [31, Chapter 10].

Lemma 2.1 *Let $\bar{x} \in X$ be a Slater point of g. Then*

$$\hat{\mu}_\kappa \in M := \left\{ \mu \in \mathbb{R}^m_+ : \|\mu\|_1 \leq \frac{f(\bar{x}) + \frac{\alpha}{2}\|\bar{x}\|^2 - f^*}{\min\limits_{1 \leq j \leq m} \{-g_j(\bar{x})\}} \right\}.$$

Proof: See [32], Section II-C. ■

If f^* is not available, any lower bound on f^* can be used in defining M, and the above construction is still valid when using such a lower bound in conjunction with any Slater point $\bar{x} \in X$. Having defined M, we see that the norm of the gradient of $\nabla_x L_\kappa(\cdot, \mu)$ can be uniformly upper-

bounded for all $\mu \in M$, and $\nabla_x L_\kappa(\cdot, \mu)$ is therefore Lipschitz. Denote its Lipschitz constant by L_p. We now define a synchronous, ensemble-level primal-dual projection method for finding $\hat{z}_\kappa$ based on [31]. It relies on the Euclidean projections onto X and M, denoted $\Pi_X[\cdot]$ and $\Pi_M[\cdot]$, respectively.

Algorithm 2.1 *Let $x(0) \in X$ and $\mu(0) \in M$ be given. For values $k = 0, 1, \ldots$, execute*

$$x(k+1) = \Pi_X \left[x(k) - \gamma \left(\nabla_x L_\kappa \left(x(k), \mu(k) \right) \right) \right]$$
$$\mu(k+1) = \Pi_M \left[\mu(k) + \rho \left(\nabla_\mu L_\kappa \left(x(k), \mu(k) \right) \right) \right].$$

$\triangle$

Here γ and ρ are stepsizes whose values will be determined in Theorems 2.1 and 2.2 in Section 2.4. Algorithm 2.1 will serve as a basis for the asynchronous algorithm developed in Section 2.3, though, as we will see, significant modifications must be made to this update law to account for asynchronous behavior in the network.

2.2 Bounds on Regularization Error

In this section we briefly cover bounds on two errors that result from the Tikhonov regularization of L. For more discussion, we refer the reader to Section 3.2 in [33] for regularized Lagrangian methods, and to Chapter 12.2 in [30] for a discussion of regularization error in general variational inequalities.

For any fixed choice of α and β, denote the corresponding solution to Problem 2.3 by $\hat{z}_{\alpha,\beta}$. It is known that as $\alpha \downarrow 0$ and $\beta \downarrow 0$ across a sequence of problems, the solutions $\hat{z}_{\alpha,\beta} \to \hat{z}_0$, where $\hat{z}_0$ is the solution to Problem 2.2 with least Euclidean norm [30, Theorem 12.2.3]. Here, we are not interested in solving a sequence of problems for evolving values of α and β because of the computational burden of doing so; instead, we solve only a single problem. It is also known that an algorithm with an iterative regularization wherein α and β tend to zero as a function of

the iteration number can also converge to $\hat{z}_0$ [34], though here it would be difficult to synchronize changes in the regularization parameters across the network. As a result, we proceed with a fixed regularization and give error bounds in terms of the regularization parameters we use.

Our focus is on selecting the parameters α and β to satisfy desired bounds on errors introduced by the regularization. First we present error bounds and then we cover how to select α and β to bound these errors by any positive constant.

2.2.1 Error Bounds

Below we use the following four constants:

$$M_f := \max_{x \in X} \|\nabla f(x)\|, \qquad M_\mu := \max_{\mu \in M} \|\mu\|, \qquad M_{g_j} := \max_{x \in X} \|\nabla g_j(x)\|, \qquad M_x := \max_{x \in X} \|x\|.$$

We first state the error in optimal cost.

Lemma 2.2 *Let Assumptions 2.1-2.4 hold. For regularization parameters $\alpha > 0$ and $\beta > 0$, the error in optimal cost incurred by regularizing L is bounded according to*

$$|f(\hat{x}_{\alpha,\beta}) - f(\hat{x})| \leq M_f M_\mu \sqrt{\frac{\beta}{2\alpha}} + \frac{\alpha}{2} M_x^2.$$

Proof: See [33, Lemma 3.3]. ■

Next we bound the constraint violation that is possible in solving Problem 2.3.

Lemma 2.3 *Let Assumptions 2.1-2.4 hold. For $\alpha > 0$ and $\beta > 0$, the constraint violation due to regularizing L is bounded according to*

$$\max\{0, g_j(\hat{x}_{\alpha,\beta})\} \leq M_{g_j} M_\mu \sqrt{\frac{\beta}{2\alpha}} \text{ for all } j \in [m].$$

Proof: See [33, Lemma 3.3]. ■

2.2.2 Selecting Regularization Parameters

We now discuss one possible choice rule for selecting α and β based upon Lemmas 2.2 and 2.3. Both lemmas suggest using $\beta < \alpha$ to achieve smaller errors and, given that we expect $\alpha < 1$, we choose $\beta = \alpha^3/2$. Suppose that there is some maximum error $\varepsilon > 0$ specified for Lemmas 2.2 and 2.3. The following result provides sufficient conditions for enforcing this bound by choosing α and β appropriately.

Lemma 2.4 *Let $\varepsilon > 0$ be given. For*

$$\hat{M} = \max\left\{\max_{j\in[m]} M_{g_j} M_\mu, M_f M_\mu\right\},$$

choosing regularization parameters

$$\alpha < \frac{2\varepsilon}{\hat{M} + M_x^2} \text{ and } \beta = \frac{\alpha^3}{2}$$

gives

$$\max\{0, g_j(\hat{x}_{\alpha,\beta})\} < \varepsilon \text{ and } |f(\hat{x}_{\alpha,\beta}) - f(\hat{x})| < \varepsilon.$$

Proof: By definition of $\hat{M}$ and Lemmas 2.2 and 2.3,

$$\max\{0, g_j(\hat{x}_{\alpha,\beta})\} \leq \hat{M}\sqrt{\frac{\beta}{2\alpha}} + M_x^2\frac{\alpha}{2}$$

for all $j \in [m]$ and

$$|f(\hat{x}_{\alpha,\beta}) - f(\hat{x})| \leq \hat{M}\sqrt{\frac{\beta}{2\alpha}} + M_x^2\frac{\alpha}{2}.$$

Thus we require $\hat{M}\sqrt{\frac{\beta}{2\alpha}} + M_x^2\frac{\alpha}{2} < \varepsilon$. Choosing $\beta = \frac{\alpha^3}{2}$ and solving for α gives the desired bound. ■

2.3 Asynchronous Optimization

In this section, we examine what happens when primal and dual updates are computed asynchronously. The agents compute primal updates and the cloud computes dual updates and, because the cloud is centralized, the dual updates in the system are computed slower than the primal updates are. Due to the difference in primal and dual update rates, we will now index the dual variable μ over the time index t, and we will continue to index the primal variable x over the time index k. In this section, we make use of the optimization framework in Sections 6.1 and 6.2 of [18]. Throughout this section, discussions will have the same value of $\mu(t)$ onboard all agents simultaneously, and this is shown to be a necessary condition for convergence in Section 2.5.

2.3.1 Per-Agent Primal Update Law

The exact update law used by agent i will be detailed below. For the present discussion, we need only to understand a few basic facts about the distribution of communications and computations in the system. Agent i will store values of some other agents' states in its onboard computer, but will only update its own state within that state vector; states stored by agent i corresponding to other agents will be updated only when those agents send their state values to agent i. Because these operations occur asynchronously, there is no reason to expect that agents i and j (with $i \neq j$) will agree upon the values of any states in the network.

As a result, we index each agent's state vector using a superscript: agent i's copy of the state of the system is denoted x^i and agent i's copy of its own state is denoted x_i^i. In this notation we say that agent i updates x_i^i but not x_j^i for any $j \neq i$. The state value x_j^i is precisely the content of messages from agent j to agent i and its value onboard agent i is changed only when agent i receives messages from agent j (and this change occurs immediately when messages are received by agent i).

To prevent unnecessary communications among the agents, we only require two agents to communicate if each needs the other's state value in its computations. We make this notion precise

in the following definition.

Definition 2.2 *Agent j is an* essential neighbor *of agent i (where $i \neq j$) if $\nabla_{x_j} L_\kappa := \frac{\partial L_\kappa}{\partial x_j}$ depends upon x_i. The set of indices of all essential neighbors of agent i is called its* essential neighborhood*, denoted $\mathcal{N}_i$.* $\lozenge$

We illustrate the role of Definition 2.2 in defining communications among the agents in the following example.

Example 2.1 *Consider a system with four agents with scalar states. For all $i \in [4]$, we have*

$$f_i(x_i) = x_i,$$

and the constraints are

$$g_1(x) = \frac{1}{2}(x_1 - x_2)^2 \text{ and } g_2(x) = \frac{1}{2}(x_3 - x_4)^2,$$

with $c \equiv 0$. For $\alpha, \beta > 0$, we find that

$$\nabla_{x_1} L_\kappa(x, \mu) = (1 + \alpha + \mu_1)x_1 - \mu_1 x_2.$$

As a result, agent 1's essential neighborhood is $\mathcal{N}_1 = \{2\}$. We also find $\mathcal{N}_2 = \{1\}$, $\mathcal{N}_3 = \{4\}$ and $\mathcal{N}_4 = \{3\}$. As a result, agents 1 and 2 need only to communicate with each other and store each other's states; neither needs to communicate with agents 3 or 4 at any point, nor to store the states of agents 3 and 4. Similarly, agents 3 and 4 communicate and store each other's states, but never communicate with agents 1 and 2 and therefore do not store their states. Agents that communicate states with each other do not need to do so simultaneously and can do so with any timing. Then at each timestep, there are four possible directed edges that can be active, and the union of all communication graphs over all timesteps is shown in Figure 2.1.

Here we see that the agents' communications need not comprise a graph which is complete, nor even one which is connected in any sense. What results then is a system in which there may be

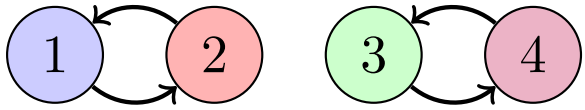

Figure 2.1: The union of all possible communication graphs over all timesteps in Example 1. This graph is neither complete, nor is it even (strongly or weakly) connected, though it provides all information exchanges necessary to use Algorithm 2.2.

multiple groups of agents which do not interact at all and which may indeed not even know of each other's existence, though they are jointly solving an optimization problem. △

Clearly $j \in \mathcal{N}_i$ if and only if $i \in \mathcal{N}_j$, and thus agent i both sends information to and receives information from its essential neighbors. While each agent only needs to store the states of its essential neighbors, we proceed as though each agent stores a full state vector in order to circumvent the need to track different dimensions of agents' states. For agent i, one can assume that x_j^i is fixed at zero for all $j \notin \mathcal{N}_i$. Rather than considering a fixed communication topology and analyzing an optimization algorithm developed over that topology, Example 2.1 shows that we take the opposite approach: the information dependencies in the system determine which agents must communicate because these dependencies define the agents' essential neighborhoods. Of course, in some cases, it will be difficult for two agents to communicate and they will do so only occasionally and without any specified schedule, and this is permitted by the asynchronous problem formulation we develop below.

The agents' primal updates also do not occur concurrently with dual updates (which will be computed by the cloud). It is therefore necessary to track which dual variable the agents currently have onboard. At time k, if agent i has $\mu(t)$ in its onboard computer, we denote agent i's copy of x by $x^i(k;t)$.

Each agent is allowed to compute its state updates using any clock it wishes, regardless of the timing of the other agents' clocks. We use the symbol K to denote a virtual global clock which contains the clock ticks of each agent's clock, and K can be understood as containing ordered indices of instants in time at which some number of agents compute state updates. Without loss of generality, we take $K = \mathbb{N}$. We denote the set of time indices at which agent i computes its state

updates[2] by K^i, i.e.,

$$K \supseteq K^i := \{k \mid x_i^i \text{ is updated by agent } i \text{ at time } k\}.$$

At times $k \in K \backslash K^i$ agent i does not compute any state updates and hence $x_i^i(k;t)$ does not change at these times, though $x_{-i}^i(k;t)$ can still change if a transmission from another agent arrives at agent i at time k. We note that K and the sets K^i need not be known by the agents as they are merely tools used in the analysis of the forthcoming asynchronous algorithm. We also take $T = \mathbb{N}$ as the set of ticks of the dual update clock in the cloud without loss of generality, though there need not be any relationship between T and K.

Suppose that agent j computes a state update at time k_a and then begins transmitting its state to agent i also at time k_a. Due to communication delays, this transmission may not arrive at agent i until, say, time $k_b > k_a$. Suppose further that agent j's next transmission to agent i does not arrive at agent i until time $k_c > k_b$. It will be useful in the following discussion to relate the time k_b (at which the first transmission arrives) to the time k_a (at which it was originally computed by agent j). Suppose at time k, with $\mu(t)$ onboard all agents, that agent i has some value of agent j's state, denoted $x_j^i(k;t)$. We use $\tau_j^i(k)$ to denote the time at which the value of $x_j^i(k;t)$ was originally computed by agent j. Above, $\tau_j^i(k_b) = k_a$, and because the value of $x_j^i(\cdot;t)$ will not change again after k_b until time k_c, we have $\tau_j^i(k') = k_a$ for all $k_b \leq k' < k_c$. We similarly define $\tau_i^c : T \to K$ for all $i \in [N]$ to fulfill the same role for transmissions of state values from agent i to the cloud: at time t in the cloud, $\tau_i^c(t)$ is the time k at which agent i computed the state value it most recently sent to the cloud[3]. For all i, j, and k we have $0 \leq \tau_j^i(k) \leq k$ by definition, and we impose the following assumption on K^i, T, τ_j^i, and τ_i^c for all $i \in [N]$ and $j \in [N]$.

Assumption 2.5 *For all $i \in [N]$ the set K^i is infinite, and for a sequence $\{k_d\}_{d=1}^{\infty}$ in K^i tending*

[2]If computing a state update takes some non-zero number of timesteps, we can make K^i the set of times at which agent i's computation of a state update completes. For simplicity we assume that computing a state update takes agent i zero time and that state updates are computed by agent i at the points in time indexed by K^i.

[3]The agents can send multiple state values to the cloud between dual updates, though only the most recent transmission from agent i to the cloud will be kept by the cloud.

to infinity we have

$$\lim_{d\to\infty} \tau_j^i(k_d) = \infty \tag{2.2}$$

for all $j \in \mathcal{N}_i$. Furthermore, the set T is infinite and for a sequence $\{t_d\}_{d=1}^{\infty}$ in T tending to infinity we have

$$\lim_{d\to\infty} \tau_i^c(t_d) = \infty \tag{2.3}$$

for all $i \in [N]$. $\lozenge$

Requiring that K^i be infinite guarantees that no agent will stop updating its state and Equation (2.2) guarantees that no agent will stop sending state updates to its essential neighbors. Similarly, T being infinite guarantees that the cloud continues to update μ and Equation (2.3) ensures that no agent stops sending its state to the cloud. Assumption 2.5 can therefore be understood as ensuring that the system "keeps running."

For a fixed $\mu(t)$, agent i's update law is written as follows (where $j \neq i$):

$$x_i^i(k+1;t) = \begin{cases} \Pi_{X_i}\Big[x_i^i(k;t) - \gamma \nabla_{x_i} L_\kappa\big(x^i(k;t), \mu(t)\big)\Big] & k \in K^i \\ x_i^i(k;t) & k \notin K^i \end{cases} \tag{2.4}$$

$$x_j^i(k+1;t) = \begin{cases} x_j^j\big(\tau_j^i(k+1);t\big) & i \text{ receives } j\text{'s state at } k+1 \\ x_j^i(k;t) & \text{otherwise} \end{cases}. \tag{2.5}$$

This update law has each agent performing gradient descent in its own state and waiting for other agents to update their states and send them to the others in the network. This captures in a precise way that agent i immediately incorporates transmissions from other agents into its current state value and that such state values will generally be "out-dated," as indicated by the presence of the τ_j^i term on the right-hand side of Equation (2.5).

2.3.2 Cloud Dual Update Law

While optimizing with $\mu(t)$ onboard, the agents compute some number of state updates using Equation (2.4) and then send their states to the cloud. It is not assumed that the agents send their states to the cloud at the same time or that they do so after the same number of state updates. Once the cloud has received states from the agents, it computes $\mu(t+1)$ and sends $\mu(t+1)$ to the agents, and then this process repeats. Assumption 2.5 specified that these operations do not cease being executed, and we impose the following basic assumption on the sequence of updates that take place in the system.

Assumption 2.6

a. When the cloud sends $\mu(t+1)$ to the agents, it arrives in finite time.

b. Any transmission originally sent from agent i to agent j while they have $\mu(t)$ onboard is only used by agent j if it is received before $\mu(t+1)$.

c. All transmissions arrive in the order in which they were sent.

d. There is an increasing sequence of times $\{k_t\}_{t\in T}$ such that only $\mu(t)$ is used in the agents' state updates at timesteps $k \in K$ satisfying $k_t \leq k < k_{t+1}$. $\Diamond$

Assumption 2.6.a is enforced simply to ensure that the optimization process does not stall and is easily satisfied in practice. Assumption 2.6.b is enforced because $\mu(t)$ parameterizes

$$\hat{x}^t := \arg\min_{x\in X} L_\kappa(x, \mu(t)),$$

which is the point the agents approach while optimizing with $\mu(t)$ onboard. Suppose that a message from agent i is sent to agent j while they have $\mu(t)$ onboard, but is received after they have $\mu(t+1)$ onboard. We will in general have $\hat{x}^t \neq \hat{x}^{t+1}$ so that the arrival of agent i's message to agent j effectively redirects agent j away from $\hat{x}^{t+1}$ and toward $\hat{x}^t$, delaying (or preventing) progress of the optimization algorithm. With respect to implementation, little needs to be done to

enforce this assumption. All communications between agents can be transmitted along with the timestamp t of the dual variable onboard the agent sending the message at the time it was sent. The agent receiving this message can compare the value of t in the message with the timestamp of the dual variable it currently has onboard, and any message with a mismatched timestamp can be discarded. Assumption 2.6.b can therefore be implemented in software without further constraining the agents' behavior or the optimization problem itself.

Assumption 2.6.c will be satisfied by a number of communication protocols, including TCP [35, Section 13], which is used on much of the internet, and does not constrain the agents' behavior because it can be enforced in software by choosing an applicable communication protocol. It is enforced here to prevent pathological behavior that can prevent the optimization process from converging at all. Assumption 2.6.d enforces that the agents use the same value of the dual variable in their updates. Assumption 2.6.d is the lone point of synchrony in the system and is necessary for convergence of the asynchronous algorithm. This necessity is verified by a counter-example in Section 2.5 wherein violating only Assumption 2.6.d causes the system not to converge.

After the agents have taken some number of steps using $\mu(t)$ and have sent their states to the cloud, the cloud aggregates these states into a vector which we denote x_t^c, defined as

$$x_t^c = \begin{pmatrix} x_1^1\big(\tau_1^c(t);t\big) \\ \vdots \\ x_N^N\big(\tau_N^c(t);t\big) \end{pmatrix}.$$

Then we adapt the dual update in Algorithm 2.1 to account for the time the cloud spends waiting to receive transmissions from the agents, giving

$$\mu(t+1) = \Pi_M\left[\mu(t) + \rho\left(g(x_t^c) - \beta\mu(t)\right)\right].$$

2.3.3 Asynchronous Primal-Dual Update Law

We now state the full asynchronous primal-dual algorithm that will be the focus of the remainder of the chapter. Below we use the notation C^i to denote the set of times at which agent i sends its state to the cloud. We also use the notation R_j^i to denote the set of times at which agent j sends its state to agent i; if $j \notin \mathcal{N}_i$, then $R_j^i = \emptyset$. Note that C^i need not have any relationship to K^i or R_j^i and that agent i need not know C^i as it is merely a tool used for analysis. Similarly, R_j^i does not need to have any relationship to K^i or C^i and does not need to be known by any agent. We state the algorithm with the cloud waiting for each agent's state before computing a dual update because this will typically be the desired behavior in a system. However, we do point out how to eliminate this assumption and the impact of this removal in Remark 2.4 in the next section.

Algorithm 2.2

Step 0: Initialize all agents and the cloud with $x(0) \in X$ and $\mu(0) \in M$. Set $t = 0$ and $k = 0$.

Step 1: For all $i \in [N]$ and all $j \in \mathcal{N}_i$, if $k \in R_j^i$, then agent j sends $x_j^j(k;t)$ to agent i (though it may not be received for some time).

Step 2: For all $i \in [N]$ and all $j \in \mathcal{N}_i$, execute

$$x_i^i(k+1;t) = \begin{cases} \Pi_{X_i}\left[x_i^i(k;t) - \gamma \nabla_{x_i} L_\kappa\big(x^i(k;t), \mu(t)\big)\right] & k \in K^i \\ x_i^i(k;t) & k \notin K^i \end{cases}$$

$$x_j^i(k+1;t) = \begin{cases} x_j^j\big(\tau_j^i(k+1);t\big) & i \text{ receives } j\text{'s state at time } k+1 \\ x_j^i(k;t) & \text{otherwise} \end{cases}.$$

Step 3: If $k+1 \in C^i$, agent i sends $x_i^i(k+1;t)$ to the cloud. Set $k := k+1$. If all components of x_t^c have been updated since the agents received $\mu(t)$, the cloud computes

$$\mu(t+1) = \Pi_M\left[\mu(t) + \rho\left(g(x_t^c) - \beta\mu(t)\right)\right]$$

and sends $\mu(t+1)$ to the agents. Set $t := t+1$.

Step 4: Return to Step 1. $\triangle$

We show in Section 2.4 that Algorithm 2.2 approximately converges to $(\hat{x}_\kappa, \hat{\mu}_\kappa)$.

2.4 Convergence of Asynchronous Primal-Dual Method

In this section we examine the convergence properties of Algorithm 2.2 and develop its convergence rates. For clarity of presentation, we first show results that assume that all agents send their states to the cloud before the cloud computes each dual update. Once the main results of this section are established, we explain how to eliminate this assumption in Remark 2.4.

2.4.1 Block Maximum Norm Basics

First we consider the agents optimizing in the primal space with a fixed $\mu(t)$. We will examine convergence using a block-maximum norm similar to that defined in Section 3.1.2 of [18]. First, for a vector $x \in X := X_1 \times \cdots \times X_N$, we can decompose x into its components as $x := (x_1, \ldots, x_N)$, and we refer to each such component of x as a *block* of x; in Algorithm 2.2, agent i updates block i of x^i. Using the notion of a block we have the following definition.

Definition 2.3 *For a vector $x \in \mathbb{R}^n$ comprised of N blocks, with the i^{th} block being $x_i \in \mathbb{R}^{n_i}$, the norm $\|\cdot\|_{2,\infty}$ is defined as*[4]

$$\|x\|_{2,\infty} = \max_{i \in [N]} \|x_i\|_2,$$

where $\|\cdot\|_2$ denotes the Euclidean norm on $\mathbb{R}^{n_i}$. $\Diamond$

We have the following lemma regarding the matrix norm induced on $\mathbb{R}^{n \times n}$ by $\|\cdot\|_{2,\infty}$. In it, we use the notion of a block of a matrix. For $n = \sum_{i=1}^N n_i$, the i^{th} block of $A \in \mathbb{R}^{n \times n}$ is the $n_i \times n$ matrix formed by rows with indices $\sum_{k=1}^{i-1} n_k + 1$ through $\sum_{k=1}^{i} n_k$ in A. We denote the i^{th} block of A by $A^{[i]}$ so that A can be reconstituted by vertically concatenating its blocks as in

[4]For concreteness we focus on the $(2, \infty)$-norm, though the results we present can be extended to some more general weighted block-maximum norms of the form $\|x\|_{max} = \max_{i \in [N]} \|x_i\|_{p_i} / w_i$, where $w_i > 0$ and $p_i \in \mathbb{N}$ for all $i \in [N]$.

$$A = \left(\begin{array}{|c|} \hline A^{[1]} \\ \hline A^{[2]} \\ \hline \vdots \\ \hline A^{[N]} \\ \hline \end{array} \right).$$

We have the following result.

Lemma 2.5 *For all* $A \in \mathbb{R}^{n \times n}$,

$$\|A\|_{2,\infty} \leq \|A\|_2.$$

Proof: For $A^{[i]}$ the i^{th} block of A, let $A^{[i]}_{\ell,j}$ be the $\ell^{th} j^{th}$ entry of that block and let $\mathbb{S}^{n-1}$ be the unit sphere in $\mathbb{R}^n$. Then for any $x \in \mathbb{S}^{n-1}$ and $i \in [N]$ we have

$$\|A^{[i]}x\|_2 = \left(\sum_{k=1}^{n_i} \left(\sum_{j=1}^{n} A^{[i]}_{k,j} x_j \right)^2 \right)^{1/2}. \tag{2.6}$$

Each term on the right-hand side of Equation (2.6) is manifestly positive so that summing over every block gives

$$\|A^{[i]}x\|_2 \leq \left(\sum_{i=1}^{N} \sum_{k=1}^{n_i} \left(\sum_{j=1}^{n} A^{[i]}_{k,j} x_j \right)^2 \right)^{1/2},$$

which we can write in terms of A as

$$\begin{aligned} \left(\sum_{i=1}^{N} \sum_{k=1}^{n_i} \left(\sum_{j=1}^{n} A^{[i]}_{k,j} x_j \right)^2 \right)^{1/2} &= \left(\sum_{\ell=1}^{n} \left(\sum_{j=1}^{n} A_{\ell,j} x_j \right)^2 \right)^{1/2} \\ &= \|Ax\|_2. \end{aligned}$$

Then, taking the maximum over all blocks, we have

$$\|Ax\|_{2,\infty} = \max_{i \in [N]} \|A^{[i]}x\|_2 \leq \|Ax\|_2,$$

for all $x \in \mathbb{S}^{n-1}$, and the result follows by taking the supremum over $x \in \mathbb{S}^{n-1}$. ■

We also have the following elementary lemma relating norms of vectors.

Lemma 2.6 *For all $x \in X$,*

$$\|x\|_2^2 \leq N \|x\|_{2,\infty}^2 \quad \text{and} \quad \|x\|_2 \leq \sqrt{N} \|x\|_{2,\infty}.$$

Proof: We find

$$\|x\|_2^2 = \sum_{i \in [N]} \|x_i\|_2^2 \leq N \max_{i \in [N]} \|x_i\|_2^2 = N \|x\|_{2,\infty}^2,$$

and then take the square root. ■

2.4.2 Convergence in the Primal Space

We now examine what happens when the agents are optimizing in between transmissions from the cloud. We consider the case of some $\mu(t) \in M$ fixed onboard all agents and, as in Section 2.3, we use $\hat{x}^t = \arg\min_{x \in X} L_\kappa\big(x, \mu(t)\big)$. Under these conditions, the agents are asynchronously minimizing the function

$$L_\kappa^t(x) := L_\kappa\big(x, \mu(t)\big)$$

until $\mu(t+1)$ arrives from the cloud. To assess the convergence of Algorithm 2.2 in the primal space, we define a sequence of sets $\{X^t(s)\}_{s \in \mathbb{N}}$ to make use of the framework in Sections 6.1 and 6.2 of [18]. These sets must satisfy the following assumption which is based on the assumptions in those sections.

Assumption 2.7 *With a fixed value of t and a fixed $\mu(t)$ onboard all agents, the sets $\{X^t(s)\}_{s \in \mathbb{N}}$ satisfy:*

a. $\cdots \subseteq X^t(s+1) \subseteq X^t(s) \subseteq \cdots \subseteq X$

b. $\lim_{s \to \infty} X^t(s) = \{\hat{x}^t\}$

c. For all i, there are sets $X_i^t(s) \subseteq X_i$ satisfying

$$X^t(s) = X_1^t(s) \times \cdots \times X_N^t(s)$$

d. For all $y \in X^t(s)$ and $i \in [N]$, $\theta_i(y) \in X_i^t(s+1)$, where $\theta_i(y) := \Pi_{X_i}[y_i - \gamma \nabla_{x_i} L_\kappa^t(y)]$. $\Diamond$

Unless otherwise noted, when writing $x \in X^t(s)$ for some vector x, the set $X^t(s)$ is chosen with the largest value of s that makes the statement true. Assumptions 2.7.a and 2.7.b require that we have a nested chain of sets to descend that ends with $\hat{x}^t$. Assumption 2.7.c allows the blocks of the primal variable to be updated independently by the agents while still guaranteeing that progress toward $\hat{x}^t$ is being made. Assumption 2.7.d guarantees forward progress down the chain of sets $\{X^t(s)\}_{s \in \mathbb{N}}$ whenever an agent computes a state update. More will be said about this assumption and its consequences below in Remark 2.1.

Recalling that L_p is the (maximum, over $\mu \in M$) Lipschitz constant of $\nabla_x L_\kappa(\cdot, \mu)$, we define the constant

$$q_p = \max\{|1 - \gamma\alpha|, |1 - \gamma L_p|\}.$$

We then have the following lemma that lets us determine the value of q_p based upon γ.

Lemma 2.7 *For $\gamma \in (0, 2/L_p)$ and $\alpha \in (0, L_p)$ we have $q_p \in (0, 1)$. Furthermore the minimum value of q_p is*

$$q_p^* = \frac{L_p - \alpha}{L_p + \alpha} \quad \text{when} \quad \gamma = \frac{2}{L_p + \alpha}.$$

Proof: See Theorem 3 on page 25 of [36]. ■

We proceed under the restrictions that $\gamma \in (0, 2/L_p)$ and $\alpha \in (0, L_p)$ for the remainder of the chapter. To simplify the presentation of results in this section, for all $t \in T$ we assume that all agents simultaneously received $\mu(t)$ at some time k_t. This k_t serves the same role as in Assumption 2.6.d, though the agents do not actually need to receive $\mu(t)$ at the exact same time and indeed any means of enforcing Assumption 2.6.d will suffice. We retain this assumption on k_t for the simplicity it provides below.

For each k_t, we define the quantity

$$D(k_t) := \max_{i \in [N]} \left\| x^i(k_t; t) - \hat{x}^t \right\|_{2,\infty},$$

which is the "worst-performing" block onboard any agent with respect to distance from $\hat{x}^t$. For a fixed value of t we define each element in the sequence of sets $\{X^t(s)\}_{s \in \mathbb{N}}$ as

$$X^t(s) = \left\{ y \in X : \left\| y - \hat{x}^t \right\|_{2,\infty} \leq q_p^s D(k_t) \right\}. \tag{2.7}$$

By definition, at time k_t we have $x^i(k_t; t) \in X^t(0)$ for all i, and moving from $X^t(0)$ to $X^t(1)$ requires contracting toward $\hat{x}^t$ (with respect to $\|\cdot\|_{2,\infty}$) by a factor of q_p. We have the following proposition.

Proposition 2.1 *The collection of sets $\{X^t(s)\}_{s \in \mathbb{N}}$ as defined in Equation* (2.7) *satisfies Assumption 2.7.*

Proof: By definition $X^t(s) \subseteq X$ for all s. From Equation (2.7), we see that

$$X^t(s+1) = \left\{ y \in X : \left\| y - \hat{x}^t \right\|_{2,\infty} \leq q_p^{s+1} D(k_t) \right\}.$$

Because $q_p \in (0, 1)$, we have $q_p^{s+1} < q_p^s$, so that for $y \in X^t(s+1)$ we find

$$\left\| y - \hat{x}^t \right\|_{2,\infty} \leq q_p^{s+1} D(k_t) < q_p^s D(k_t),$$

giving $y \in X^t(s)$ as well. Then $X^t(s+1) \subseteq X^t(s) \subseteq X$ for all $s \in \mathbb{N}$, and Assumption 2.7.a is satisfied.

We see that

$$\begin{aligned} \lim_{s \to \infty} X^t(s) &= \lim_{s \to \infty} \left\{ y \in X : \left\| y - \hat{x}^t \right\|_{2,\infty} \leq q_p^s D(k_t) \right\} \\ &= \left\{ y \in X : \left\| y - \hat{x}^t \right\|_{2,\infty} \leq 0 \right\} = \{\hat{x}^t\}, \end{aligned}$$

which follows because $\|\cdot\|_{2,\infty}$ is a norm. Then Assumption 2.7.b is satisfied as well.

For Assumption 2.7.c, the definition of $\|\cdot\|_{2,\infty}$ lets us easily decompose $X^t(s)$. In particular, we see that

$$\left\|y - \hat{x}^t\right\|_{2,\infty} \leq q_p^s D(k_t) \text{ if and only if } \|y_i - \hat{x}_i^t\|_2 \leq q_p^s D(k_t)$$

for all $i \in [N]$. Immediately then we have

$$X_i^t(s) = \left\{y_i \in X_i : \|y_i - \hat{x}_i^t\|_2 \leq q_p^s D(k_t)\right\}$$

from which it is clear that $X^t(s) = X_1^t(s) \times \cdots \times X_N^t(s)$ and thus that Assumption 2.7.c is satisfied.

Finally, we show that Assumption 2.7.d is satisfied. For a fixed t and fixed $\mu(t)$ take some $y \in X^t(s)$. Recall the following exact expansion of $\nabla_x L_\kappa^t$:

$$\begin{aligned}
\nabla_x L_\kappa^t(y) - \nabla_x L_\kappa^t(\hat{x}^t) &= \int_0^1 \nabla_x^2 L_\kappa^t\big(\hat{x}^t + \tau(y - \hat{x}^t)\big)(y - \hat{x}^t) d\tau \\
&= \left(\int_0^1 \nabla_x^2 L_\kappa^t\big(\hat{x}^t + \tau(y - \hat{x}^t)\big) d\tau\right) \cdot (y - \hat{x}^t) \\
&=: H_\kappa^t(y)(y - \hat{x}^t), \qquad (2.8)
\end{aligned}$$

where we have defined H_κ^t as

$$H_\kappa^t(y) := \int_0^1 \nabla_x^2 L_\kappa^t\big(\hat{x}^t + \tau(y - \hat{x}^t)\big) d\tau.$$

Using the non-expansive property of $\Pi_{X_i}[\cdot]$ with respect to $\|\cdot\|_2$, for any $y \in X^t(s)$ we have

$$\begin{aligned}
\|\theta_i(y) - \hat{x}_i^t\|_2 &= \left\|\Pi_{X_i}\left[y_i - \gamma\nabla_{x_i}L_\kappa^t(y)\right] - \Pi_{X_i}\left[\hat{x}_i^t - \gamma\nabla_{x_i}L_\kappa^t(\hat{x}^t)\right]\right\|_2 \\
&\leq \|y_i - \gamma\nabla_{x_i}L_\kappa^t(y) - \hat{x}_i^t + \gamma\nabla_{x_i}L_\kappa^t(\hat{x}^t)\|_2 \\
&\leq \max_{i\in[N]} \|y_i - \gamma\nabla_{x_i}L_\kappa^t(y) - \hat{x}_i^t + \gamma\nabla_{x_i}L_\kappa^t(\hat{x}^t)\|_2 \\
&= \left\|y - \hat{x}^t - \gamma\left(\nabla_x L_\kappa^t(y) - \nabla_x L_\kappa^t(\hat{x}^t)\right)\right\|_{2,\infty} \\
&= \left\|y - \hat{x}^t - \gamma H_\kappa^t(y)(y - \hat{x}^t)\right\|_{2,\infty} \\
&= \left\|(I - \gamma H_\kappa^t(y))(y - \hat{x}^t)\right\|_{2,\infty} \\
&\leq \left\|I - \gamma H_\kappa^t(y)\right\|_{2,\infty} \left\|y - \hat{x}^t\right\|_{2,\infty} \\
&\leq \|I - \gamma H_\kappa^t(y)\|_2 \left\|y - \hat{x}^t\right\|_{2,\infty}, \qquad (2.9)
\end{aligned}$$

where the third equality follows from Equation (2.8) and where the last inequality follows from Lemma 2.5. For any choice of $\mu(t) \in M$, the α-strong monotonicity and L_p-Lipschitz properties of $\nabla_x L_\kappa(\cdot, \mu(t))$ give $\alpha I \preceq H_\kappa^t(\cdot) \preceq L_p I$, which implies that the eigenvalues of $H_\kappa^t(\cdot)$ are bounded above by L_p and below by α for all $\mu(t) \in M$. Using this fact and that $H_\kappa^t(y)$ is symmetric, we see that

$$\begin{aligned}
\|I - \gamma H_\kappa^t(y)\|_2 &= \max\left\{|\lambda_{min}(I - \gamma H_\kappa^t(y))|, |\lambda_{max}(I - \gamma H_\kappa^t(y)|\right\} \\
&= \max\{|1 - \gamma\alpha|, |1 - \gamma L_p|\} \\
&= q_p,
\end{aligned}$$

where λ_{min} and λ_{max} denote the minimum and maximum eigenvalues of a matrix, respectively. Then from Equation (2.9), and the fact that $\left\|y - \hat{x}^t\right\|_{2,\infty} \leq q_p^s D(k_t)$ by hypothesis, we find

$$\|\theta_i(y) - \hat{x}_i^t\|_2 \leq q_p \left\|y - \hat{x}^t\right\|_{2,\infty} \leq q_p^{s+1} D(k_t),$$

so that $\theta_i(y) \in X_i^t(s+1)$ as desired. ■

We comment on one consequence of Assumption 2.7 in particular below where we use the notation

$$X^t_{-i}(s) := X^t_1(s) \times \cdots \times X^t_{i-1}(s) \times X^t_{i+1}(x) \times \cdots \times X^t_N(s).$$

Remark 2.1 *Suppose at time k agent i has state vector $x^i(k;t) \in X^t(s)$ and $k+1 \in K^i$. Then Assumption 2.7.d implies that $x^i_i(k+1;t) = \theta_i(x^i(k;t)) \in X^t_i(s+1)$. Suppose, before any other agent transmits an updated state to agent i, that agent i performs another update of its own state. Just before the second update, $x^i(k+1;t)$ is equal to $x^i(k;t)$ with the entry for x^i_i replaced with the update just computed, $\theta_i(x^i(k;t))$; all other entries of $x^i(k+1;t)$ remain unchanged from $x^i(k;t)$. Because no other agents' states have changed, we still have $x^i_{-i}(k+1;t) \in X^t_{-i}(s)$ and, as a result, $x^i(k+1;t) \in X^t(s)$. In general, $x^i(k+1;t) \notin X^t(s+1)$ here precisely because $x^i_{-i}(k+1;t)$ has not changed. Due to the fact that $x^i(k+1;t) \in X^t(s)$, the second update performed by agent i results in $\theta_i(x^i(k+1;t)) \in X^t_i(s+1)$ once more, though, in general, no further progress, e.g., to $X^t_i(s+2)$, can be made without further updates from the other agents. Then while an agent is waiting for updates from other agents, its progress toward $\hat{x}^t$ can be halted, though it does not "regress" backwards from, say, $X^t_i(s)$ to $X^t_i(s-1)$.* $\Diamond$

We proceed to use Assumption 2.7 to estimate the primal convergence rate of Algorithm 2.2.

2.4.3 Single-Cycle Primal Convergence Rate Estimate

The structure of the sets $\{X^t(s)\}_{s\in\mathbb{N}}$ enables us to extract a convergence rate estimate in the primal space. To demonstrate this point, consider the starting point of the algorithm: all agents have onboard some state $x(0) \in X^0(0)$ and dual vector $\mu(0) \in M$. Suppose that agent i takes a single gradient descent step, say at time $k^i \in K^i$, from $x(0)$ with $\mu(0)$ held fixed. From Assumption 2.7.d, this results in agent i having $x^i_i(k^i;0) = \theta_i\big(x^i(0;0)\big) = \theta_i\big(x(0)\big) \in X^0_i(1)$. Once agent i transmits the state $x^i_i(k^i;0)$ to its essential neighbors, and once all other agents themselves have taken a descent step and transmitted their states to their essential neighbors, say, at time $\bar{k}$, agent i will have $x^i(\bar{k};0) \in X^0(1)$. Agent i's next descent step, say at time $\ell > \bar{k}$, then results in $x^i_i(\ell;0) =$

$\theta_i\big(x^i(\bar{k};0)\big) \in X_i^0(2)$. Then the process of communicating and descending repeats. To keep track of how many times this process repeats, we have the following definition.

Definition 2.4 *After the agents all have just received $\mu(t)$, the first* cycle *ends as soon as (i) each agent has computed a state update and (ii) each agent has sent that updated state to all of its essential neighbors and it has been received by them. Subsequent cycles are completed when the preceding cycle has ended and criteria (i) and (ii) are met again, with any number of cycles possible between k_t and k_{t+1}.* $\Diamond$

It is possible for one agent to compute and share several state updates with the other agents within one cycle if some other agent in the network is updating more slowly. For a fixed value of $\mu(0)$ onboard all agents and a common initial state $x(0)$, the first cycle will move each agent's copy of the ensemble state from $X^0(0)$ to $X^0(1)$, the second cycle will move it from $X^0(1)$ to $X^0(2)$, etc. When the agents have $\mu(t)$ onboard, we use $c(t)$ to denote the number of cycles the agents complete before the first agent sends its state to the cloud for use in computing $\mu(t+1)$. We see that with $\mu(0)$ onboard, the agents complete $c(0)$ cycles to reach the set $X^0(c(0))$, and the cloud therefore uses an element of $X^0(c(0))$ to compute $\mu(1)$. In particular, using Assumption 2.7.d and the construction of the sets $\{X^0(s)\}_{s\in\mathbb{N}}$, this means that the convergence rate is geometric in the number of cycles completed:

$$\left\|x_0^c - \hat{x}^0\right\|_{2,\infty} \leq q_p^{c(0)} D(k_0) = q_p^{c(0)} \left\|x(0) - \hat{x}^0\right\|_{2,\infty}.$$

Crucially, it need not be the case that all agents have the same state at the beginning of each cycle for this rate estimate to apply. We show this in deriving a general primal convergence rate in the following lemma.

Lemma 2.8 *Let Assumptions 2.1-2.7 hold, let $\kappa = (\alpha, \beta)$ be fixed, and let $\gamma \in (0, 2/L_p)$. When the agents are all optimizing with $\mu(t)$ onboard and the first agent sends its state to the cloud after $c(t)$ cycles, we have*

$$\left\|x_t^c - \hat{x}^t\right\|_{2,\infty} \leq q_p^{c(t)} D(k_t). \tag{2.10}$$

Proof: Suppose the agents just received $\mu(t)$ from the cloud. For all $i \in [N]$ we have $x^i(k_t;t) \in X^t(0)$ from the definition of $D(k_t)$. Then when agent i computes a state update the result is $\theta_i\big(x^i(k_t;t)\big) \in X_i^t(1)$. When the agents have completed one cycle after receiving $\mu(t)$, say by time $\bar{k}$, we have $x^i(\bar{k};t) \in X^t(1)$. Iterating this process, after $c(t)$ cycles agent i's copy of the ensemble state moves from $X^t(0)$ to $X^t(c(t))$ for all $i \in [N]$. Then, when the agents send their states to the cloud, agent i sends an element of $X_i^t(c(t))$. Then we have $x_t^c \in X^t(c(t))$ and, by definition, $\|x_t^c - \hat{x}^t\|_{2,\infty} \leq q_p^{c(t)} D(k_t)$. ■

One can impose further assumptions, e.g., that a cycle occurs every B ticks of K, in which case the exponent of q_p in Equation (2.10) becomes $\lfloor (k_{t+1} - k_t)/B \rfloor$, though for generality we do not do so.

2.4.4 Overall Convergence Rates

Towards providing a convergence rate estimate in the dual space, we first present the following lemma.

Lemma 2.9 *For any primal-dual pairs $(x_1, \mu_1) \in X \times M$ and $(x_2, \mu_2) \in X \times M$ such that*

$$x_1 = \arg\min_{x \in X} L_\kappa(x, \mu_1) \text{ and } x_2 = \arg\min_{x \in X} L_\kappa(x, \mu_2),$$

we have

$$(\mu_2 - \mu_1)^T (g(x_1) - g(x_2)) \geq \frac{\alpha}{M_g^2} \|g(x_1) - g(x_2)\|^2.$$

In addition,

$$\|\mu_1 - \mu_2\| \geq \frac{\alpha}{M_g} \|x_1 - x_2\|.$$

Proof: See [33, Lemma 4.1]. ■

We now prove approximate convergence in the dual space and estimate the rate of convergence there.

Theorem 2.1 *Let all hypotheses of Lemma 2.8 hold and let the dual step-size satisfy*

$$0 < \rho < \rho_0 := \min\left\{\frac{2\alpha}{M_g^2 + 2\alpha\beta}, \frac{2\beta}{1+\beta^2}\right\},$$

where $M_g = \max_{x\in X} \|\nabla g(x)\|$. *Then for all* $t \geq 0$ *in Algorithm 2.2*

$$\|\mu(t+1) - \hat{\mu}_\kappa\|^2 \leq q_d^{t+1}\|\mu(0) - \hat{\mu}_\kappa\|^2 + \sum_{\ell=0}^{t} q_d^{t-\ell}\Big(q_d N M_g^2 L_x^2 q_p^{2c(\ell)} + 2\sqrt{N}\rho^2 M_g^2 L_x D_x q_p^{c(\ell)}\Big),$$

where $(0,1) \ni q_d := (1-\rho\beta)^2 + \rho^2$, $D_x := \max_{x,y\in X}\|x-y\|$ *is the diameter of* X, *and* $L_x := \max_{i\in[N]}\max_{x_i,y_i\in X_i}\|x_i - y_i\|$ *is the maximum diameter among the sets* X_i, $i \in [N]$.

Proof: Using the non-expansive property of the projection operator $\Pi_M[\cdot]$ and expanding we find

$$\begin{aligned}\|\mu(t+1) - \hat{\mu}_\kappa\|^2 &= \left\|\Pi_M\left[\mu(t) + \rho\left(g(x_t^c) - \beta\mu(t)\right)\right] - \Pi_M\left[\hat{\mu}_\kappa + \rho\left(g(\hat{x}_\kappa) - \beta\hat{\mu}_\kappa\right)\right]\right\|^2 \\ &\leq (1-\rho\beta)^2\|\mu(t) - \hat{\mu}_\kappa\|^2 + \rho^2\|g(\hat{x}_\kappa) - g(x_t^c)\|^2 \\ &\qquad - 2\rho(1-\rho\beta)\big(\mu(t) - \hat{\mu}_\kappa\big)^T\big(g(\hat{x}_\kappa) - g(x_t^c)\big).\end{aligned}$$

Adding $g(\hat{x}^t) - g(\hat{x}^t)$ inside the last set of parentheses, expanding, and applying Lemma 2.9 then gives

$$\begin{aligned}\|\mu(t+1) - \hat{\mu}_\kappa\|^2 &\leq (1-\rho\beta)^2\|\mu(t) - \hat{\mu}_\kappa\|^2 + \rho^2\|g(\hat{x}_\kappa) - g(x_t^c)\|^2 \\ &\quad - 2\rho(1-\rho\beta)\frac{\alpha}{M_g^2}\|g(\hat{x}_\kappa) - g(\hat{x}^t)\|^2 - 2\rho(1-\rho\beta)\big(\mu(t) - \hat{\mu}_\kappa\big)^T\big(g(\hat{x}^t) - g(x_t^c)\big).\end{aligned} \tag{2.11}$$

Next, we have

$$0 \leq \|(1-\rho\beta)(g(\hat{x}^t) - g(x_t^c)) + \rho(\mu(t) - \hat{\mu}_\kappa)\|^2,$$

where expanding and re-arranging gives

$$-2\rho(1-\rho\beta)(\mu(t)-\hat{\mu}_\kappa)^T(g(\hat{x}^t)-g(x_t^c)) \leq$$
$$(1-\rho\beta)^2\|g(\hat{x}^t)-g(x_t^c)\|^2+\rho^2\|\mu(t)-\hat{\mu}_\kappa\|^2. \quad (2.12)$$

Substituting Equation (2.12) into Equation (2.11) then gives

$$\|\mu(t+1)-\hat{\mu}_\kappa\|^2 \leq \left((1-\rho\beta)^2+\rho^2\right)\|\mu(t)-\hat{\mu}_\kappa\|^2+\rho^2\|g(\hat{x}_\kappa)-g(x_t^c)\|^2 \quad (2.13)$$
$$-2\rho(1-\rho\beta)\frac{\alpha}{M_g^2}\|g(\hat{x}_\kappa)-g(\hat{x}^t)\|^2+(1-\rho\beta)^2\|g(\hat{x}^t)-g(x_t^c)\|^2.$$

Next, we see that

$$\begin{aligned}\|g(\hat{x}_\kappa)-g(x_t^c)\|^2 &= \|g(\hat{x}_\kappa)-g(\hat{x}^t)+g(\hat{x}^t)-g(x_t^c)\|^2 \\ &\leq \|g(\hat{x}_\kappa)-g(\hat{x}^t)\|^2+\|g(\hat{x}^t)-g(x_t^c)\|^2 \\ &\quad +2\|g(\hat{x}_\kappa)-g(\hat{x}^t)\|\|g(\hat{x}^t)-g(x_t^c)\|, \end{aligned} \quad (2.14)$$

and substituting Equation (2.14) into Equation (2.13) gives

$$\|\mu(t+1)-\hat{\mu}_\kappa\|^2 \leq \left((1-\rho\beta)^2+\rho^2\right)\|\mu(t)-\hat{\mu}_\kappa\|^2+\left(\rho^2-2\rho(1-\rho\beta)\frac{\alpha}{M_g^2}\right)\|g(\hat{x}_\kappa)-g(\hat{x}^t)\|^2$$
$$+\left((1-\rho\beta)^2+\rho^2\right)\|g(\hat{x}^t)-g(x_t^c)\|^2+2\rho^2\|g(\hat{x}_\kappa)-g(\hat{x}^t)\|\|g(\hat{x}^t)-g(x_t^c)\|. \quad (2.15)$$

Taking $\rho \in (0, \rho_0)$, we find that

$$\rho^2-2\rho(1-\rho\beta)\frac{\alpha}{M_g^2}<0 \quad (2.16)$$

and

$$(0,1) \ni q_d := (1-\rho\beta)^2+\rho^2.$$

Substituting Equation (2.16) into Equation (2.15), and using the Lipschitz property of g, we find

$$\|\mu(t+1) - \hat{\mu}_\kappa\|^2 \leq q_d \|\mu(t) - \hat{\mu}_\kappa\|^2 + q_d M_g^2 \|\hat{x}^t - x_t^c\|^2 + 2\rho^2 M_g^2 D_x \|\hat{x}^t - x_t^c\|. \tag{2.17}$$

Lemmas 2.6 and 2.8 imply that

$$\|\hat{x}^t - x_t^c\|_2 \leq \sqrt{N} \left\|\hat{x}^t - x_t^c\right\|_{2,\infty} \leq \sqrt{N} q_p^{c(t)} L_x.$$

Using this in Equation (2.17) gives

$$\|\mu(t+1) - \hat{\mu}_\kappa\|^2 \leq q_d \|\mu(t) - \hat{\mu}_\kappa\|^2 + q_d N M_g^2 L_x^2 q_p^{2c(t)} + 2\sqrt{N}\rho^2 M_g^2 L_x D_x q_p^{c(t)},$$

where the result follows by summing over t. ■

Remark 2.2 *Theorem 2.1 shows that convergence in the dual space is governed by three terms. The first term decays as q_d^{t+1} and represents a contraction toward $\hat{\mu}_\kappa$. The next two terms are essentially error terms that result from x_t^c not equaling $\hat{x}^t$; to see this, note that an exact dual method would have $c(t) = \infty$, causing the sum in Theorem 2.1 to vanish, leaving only the contracting term. We see that larger values of $c(t)$ lead x_t^c closer to $\hat{x}^t$, causing the algorithm to approximate an ordinary dual algorithm, thus leading to smaller errors.*

In addition, the $q_d^{t-\ell}$ term outside the sum indicates that past errors contribute less to the overall dual error, with old error terms accumulating powers of q_d over time. To make faster progress using the asynchronous algorithm, one can have the agents perform small numbers of cycles for small values of t and then increase $c(t)$ as t becomes large. Such a strategy makes later error terms small while weighting earlier error terms only minimally, giving a small overall error. ◊

We now present a result on primal convergence in Algorithm 2.2.

Theorem 2.2 *Let the primal step-size $\gamma \in (0, 2/L_p)$ and let all hypotheses of Lemma 2.8 hold. Then for the sequence of primal vectors aggregated by the cloud in Algorithm 2.2, $\{x_t^c\}_{t\in\mathbb{N}}$, we*

have

$$\|x_t^c - \hat{x}_\kappa\|_2 \leq q_p^{c(t)}\sqrt{N}L_x + \frac{M_g}{\alpha}\|\mu(t) - \hat{\mu}_\kappa\|_2.$$

Proof: Adding $\hat{x}^t - \hat{x}^t$ and using Lemmas 2.6, 2.8, and 2.9 we find

$$\begin{aligned}\|x_t^c - \hat{x}_\kappa\| &\leq \|x_t^c - \hat{x}^t\| + \|\hat{x}^t - \hat{x}_\kappa\| \\ &\leq \sqrt{N}q_p^{c(t)}L_x + \frac{M_g}{\alpha}\|\mu(t) - \hat{\mu}_\kappa\|,\end{aligned}$$

where we have bounded $D(k_t)$ by L_x. ■

Convergence in the primal space is then governed by two terms, one of which behaves like a contraction whose exponent is $c(t)$ and the other which is a constant multiple of the dual error, and we again find that completing more cycles improves accuracy. We also have the following tradeoff between speed and accuracy induced by the regularization of L.

Remark 2.3 *Theorem 2.2 and Lemmas 2.4, 2.7, and 2.8 together reveal a fundamental tradeoff between convergence rate and accuracy in the primal space. On the one hand, Lemma 2.4 shows that smaller values of α lead to smaller errors while larger values of α lead to larger errors. On the other hand, Lemma 2.7 shows that larger values of α lead to smaller values of q_p, and both Lemma 2.8 and Theorem 2.2 show that smaller values of q_p lead to faster convergence through the primal space, while smaller values of α cause q_p to approach the value 1, thereby slowing convergence. Then smaller values of α lead to smaller errors at the expense of slower convergence, while larger values of α cause the system to converge more quickly, but to a point that is further away from $(\hat{x}_\kappa, \hat{\mu}_\kappa)$.*

A similar tradeoff applies to β as well: Lemmas 2.2 and 2.3 show that smaller values of β can lead to smaller errors, though the definition of q_d in Theorem 2.1 shows that a larger value of β decreases q_d, leading to faster convergence. The appropriate balance of convergence speed and accuracy of a solution depends upon the problem being solved, though, taken together, these results give one the tools to quantitatively balance these two objectives.

One can also see the use of regularizing L in Theorems 2.1 and 2.2. If one were to set $\alpha = 0$, then we would find $q_p = 1$ and primal updates would not make any progress toward $\hat{x}^t$ in Lemma 2.8. Such a case would also cause the construction of the sets $\{X^t(s)\}_{s\in\mathbb{N}}$ to break down as no "descent" down this sequence of sets could be shown. Similarly, if one were to set $\beta = 0$, we would find $q_d = 1 + \rho^2$, in which case the only way to avoid moving away *from $\hat{\mu}_\kappa$ in the dual space would be to set $\rho = 0$, thereby forestalling all progress in the dual space. Through their roles in determining q_p and q_d (and the use of these constants in the convergence analysis presented), it is evident that regularizing with α and β is essential to the analysis presented here.* $\Diamond$

We now point out how to formulate convergence rate estimates without having each agent send a state update to the cloud before it computes each dual update.

Remark 2.4 *If one allows the cloud to compute dual updates before receiving a state update from each agent, then Lemma 2.8 should be modified to account for only some values of x_t^c changing from x_{t-1}^c. In particular, if $N(t)$ agents send state updates to the cloud before it computes $\mu(t+1)$ and $M(t) := N - N(t)$ do not, we find*

$$\|x_t^c - \hat{x}^t\|_2 \leq \sqrt{N(t)q_p^{c(t)}D(k_t) + M(t)L_x}.$$

Propagating this through Theorems 2.1 and 2.2 gives overall primal and dual convergence rate estimates for this case as well. In doing so, one finds that executing a cloud update without a state update from each agent can significantly harm convergence and it will usually be preferred to have the cloud wait until it has received state information from all agents before each dual update. $\Diamond$

2.5 Non-Convergence of the Asynchronous Dual Case

In this section we provide a counterexample to show that Assumption 2.6.d is necessary for the convergence of Algorithm 2.2. In it, we allow the agents to have different values of the system's

dual variable and show that these differences can cause the primal and dual trajectories in Algorithm 2.2 not to converge at all. As will be shown, this is true even when each agent receives the most recent dual value at regular intervals and when the agents keep their states synchronized at all times.

The problem consists of two agents with scalar states and per-agent objectives

$$f_1(x_1) = 0.1x_1 \text{ and } f_2(x_2) = -0.1x_2,$$

coupling cost $c \equiv 0$, and the constraint

$$g(x) = \frac{1}{2}(x_1 - x_2)^2 - 0.2 \leq 0.$$

The regularization parameters were chosen to be $\alpha = \beta = 0.01$, giving

$$L_\kappa(x, \mu) = 0.1x_1 - 0.1x_2 + \mu\left(\frac{1}{2}(x_1 - x_2)^2 - 0.2\right) + \frac{0.01}{2}(x_1^2 + x_2^2) - \frac{0.01}{2}\mu^2,$$

along with

$$\begin{aligned}
\frac{\partial L_\kappa}{\partial x_1} &= 0.1 + \mu(x_1 - x_2) + 0.01x_1, \\
\frac{\partial L_\kappa}{\partial x_2} &= -0.1 + \mu(x_2 - x_1) + 0.01x_2, \\
\frac{\partial L_\kappa}{\partial \mu} &= \frac{1}{2}(x_1 - x_2)^2 - 0.2 - 0.01\mu.
\end{aligned}$$

Both agents are confined to the interval $[0, 5]$, giving $X = [0, 5]^2$.

In this example, we will sometimes have one agent using an old dual value for some period of time, and we denote this value by μ_{old}; its value only changes when we write $\mu_{old} \leftarrow \mu$ in the pseudocode in Algorithm 2.3. Otherwise, μ_{old} does not update with μ. Similarly, the values of x_1 and x_2 only change when explicitly updated below and operations listed sequentially below actually occur sequentially so that the agents are updating at different times. To highlight the

Algorithm 2.3 Asynchronous Dual Counterexample

Initialize $\mu \leftarrow 0$, $\mu_{old} \leftarrow 0$, $x_1 \leftarrow 0$, and $x_2 \leftarrow 0$.
for $\tau_{outer} = 1$ to 10 **do**
  for $\tau_1 = 1$ to 500 **do** % *Mode 1*
    $x_2 \leftarrow \theta_2(x_1, x_2, \mu_{old})$
    $x_1 \leftarrow \theta_1(x_1, x_2, \mu)$
    $\mu \leftarrow \theta_M(x_1, x_2, \mu)$
  end for
  $\mu_{old} \leftarrow \mu$
  for $\tau_2 = 1$ to 1500 **do** % *Mode 2*
    while $|x_1 - \theta_1(x_1, x_2, \mu)| > 10^{-5}$ **do**
      $x_1 \leftarrow \theta_1(x_1, x_2, \mu)$
    end while
    while $|x_2 - \theta_2(x_1, x_2, \mu_{old})| > 10^{-5}$ **do**
      $x_2 \leftarrow \theta_2(x_1, x_2, \mu_{old})$
    end while
    $\mu \leftarrow \theta_M(x_1, x_2, \mu)$
  end for
  $\mu_{old} \leftarrow \mu$
end for

impact of asynchrony in the dual variable, each agent always uses the most recent state of the other agent in its computations, i.e., $x_2^1 = x_2^2$ and $x_1^2 = x_1^1$. Then there is no disagreement about state values in the network and superscript indices are therefore omitted. For clarity, we write each argument of θ_1 and θ_2 out explicitly, including specifying which dual variable is being used. To simplify notation, timestamps are omitted in the pseudocode in Algorithm 2.3.

We have $L_p \approx 50.014$ so that using the stepsize bounds in Theorems 2.1 and 2.2 we can select any

$$\gamma \in \left(0, \frac{2}{50.014}\right) \approx (0, 0.03999),$$

and we choose $\gamma = 0.002$. For ρ, we find that we must satisfy

$$\rho < \min\left\{\frac{0.02}{50 + 2 \cdot 10^{-4}}, \frac{0.02}{1 + 10^{-4}}\right\} \approx 0.00039,$$

which we do by selecting $\rho = 0.0003$. This example consists of alternating between two modes, shown in Algorithm 2.3 where the dual update law in the cloud is represented by the symbol θ_M.

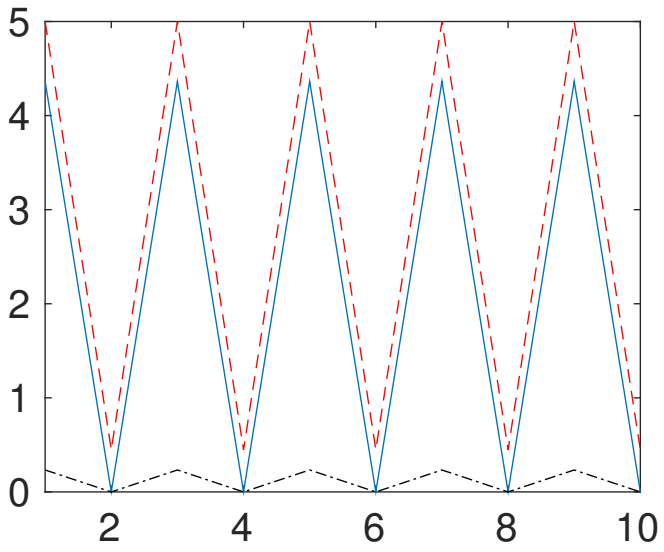

Figure 2.2: Primal and dual trajectories resulting from a simulation of Algorithm 2.3, with x_1 the upper solid line, x_2 the upper dashed line, and μ the lower dash-dotted line. All terms are plotted at the end of each iteration of the outer loop. These oscillations are of constant magnitude and do not decay, indicating that the dual variable must be synchronized across agents in Algorithm 2.2.

2.5.1 Analysis of Oscillations

Oscillations are shown in Figure 2.2 where we plot the primal and dual trajectories of a simulation implementing Algorithm 2.3. Both states and the dual variable oscillate in a non-decaying fashion, indicating that Algorithm 2.3 is not converging at all. We note here that synchronizing the dual variable in this example does indeed lead to convergence, indicating that the asynchrony of the dual values is the source of oscillations and that Assumption 2.6.d is a necessary condition for convergence of any implementation of Algorithm 2.2.

To understand why Algorithm 2.3 does not converge and why disagreements in the dual space are harmful, we briefly analyze the cause of oscillations in Algorithm 2.3 by analyzing the effects of each mode upon the state of the network. If $\mu_{old} = \mu > 0$ when starting Mode 1, then we find

that agent 2 moves towards

$$\begin{aligned}\hat{x}_2(\mu) &= \frac{0.1 + \mu x_1}{0.01 + \mu} \\ &= \frac{0.1}{0.01 + \mu} + \frac{\mu x_1}{0.01 + \mu} \\ &\approx \frac{0.1}{0.01 + \mu} + x_1,\end{aligned}$$

and agent 1 moves towards

$$\begin{aligned}\hat{x}_1(\mu) &= \frac{-0.1 + \mu x_2}{0.01 + \mu} \\ &= \frac{-0.1}{0.01 + \mu} + \frac{\mu x_2}{0.01 + \mu} \\ &\approx -\frac{0.1}{0.01 + \mu} + x_2.\end{aligned}$$

As a result, when beginning Mode 1 with $\mu_{old} = \mu > 0$, we expect x_2 to move toward x_1 a small amount and x_1 to move toward x_2 a small amount, though their positions will only change slightly due to the constraints and, as a result, μ will only see small changes.

If $\mu_{old} = \mu = 0$ when starting Mode 2, then the first loop in Mode 2 results in agent 1 moving toward the point

$$\hat{x}_1(0) = \frac{-0.1}{0.01} = -10,$$

which is projected onto X so that

$$x_1 = 0.$$

Similarly, we find that x_2 moves toward the point

$$\hat{x}_2(0) = \frac{0.1}{0.01} = 10,$$

which is projected onto X so that

$$x_2 = 5.$$

Then we expect entering Mode 2 with $\mu_{old} = \mu = 0$ will cause the agents to move apart in the first iteration. In fact, each `while` loop inside Mode 2 is run enough times that the agents arrive at $x_1 = 0$ and $x_2 = 5$ by the end of the first iteration of Mode 2. Agent 2 has μ_{old} throughout the duration of this mode so that it continues to stay at 5 each time this loop is repeated. As a result, the distance between agents 1 and 2 will make $g(x_1, x_2) > 0$, causing μ to increase when it is updated. Because only agent 1 receives the most recent dual value in this mode, x_1 will increase towards x_2 in order to satisfy g. Then starting this mode with $\mu = 0$ ends with x_2 at 5, x_1 at distance $\sqrt{2 \cdot 0.2}$ from x_2, the maximum distance allowed by g, and $\mu > 0$.

If $\mu_{old} = \mu > 0$ when starting Mode 2, the first `while` loop in Mode 2 results in agent 1 moving to

$$\begin{aligned}\hat{x}_1(\mu) &= \frac{-0.1 + \mu x_2}{\mu + 0.01} \\ &\approx -\frac{0.1}{\mu + 0.01} + x_2,\end{aligned}$$

and the second `while` loop in Mode 2 results in agent 2 moving to

$$\begin{aligned}\hat{x}_2(\mu_{old}) &= \frac{0.1 + \mu_{old} x_1}{\mu_{old} + \alpha} \\ &\approx \frac{0.1}{\mu_{old} + 0.01} + x_1.\end{aligned}$$

Then in starting Mode 2 with $\mu_{old} = \mu > 0$, we expect the agents to move toward each other and the distance between them to become small. When μ is updated at the end of this mode, μ will decrease as a result of the agents being close. After enough iterations of this mode, μ will become zero, allowing agent 1 to decrease to zero, as happened when Mode 2 was started with $\mu = 0$. Meanwhile, because μ_{old} stays at a fixed value, x_2 will continue to approach x_1 to satisfy g. This guarantees that x_2 "follows" x_1 as x_1 moves, keeping $(x_1 - x_2)^2$ small and causing μ to decrease and eventually reach zero.

Algorithm 2.3 oscillates because it starts with Mode 1 and $\mu = 0$, ending with $x_1 = 0$, $x_2 > 0$

small (because only 500 loop iterations are executed), and $\mu = 0$. Then Mode 2 is entered which causes x_2 to increase toward 5, μ to become positive, and x_1 to "chase" x_2 in order to satisfy the constraints. Then Mode 1 is entered with $\mu > 0$ where all values in the network change only by small amounts, again due to only 500 iterations being run. Then Mode 2 is entered with $\mu > 0$ which ends with the agents close and $\mu = 0$. Then this process repeats, with persistent oscillations.

Having verified the necessity of Assumption 2.6.d, the next section simulates Algorithm 2.2 with all assumptions enforced.

2.6 Simulation Results

We now present simulation results for Algorithm 2.2. We first discuss the problem to be solved and then cover our implementation. We then present numerical results that demonstrate convergence of Algorithm 2.2 on the cloud-based system and the tradeoff between convergence rate and accuracy that is induced by the Tikhonov regularization of L.

2.6.1 Problem Overview

We consider a problem of routing $N = 8$ flows through a network consisting of 8 nodes and 9 edges, representing, e.g., traffic flow or sending data across a communication network, and each agent's decision variable is the flow rate of its data through the network, which is depicted in Figure 2.3. The nodes of the network are not the agents themselves, but, instead, the agents are users of the network attempting to route traffic between certain pairs of these nodes. The starting points, ending points, and edges which comprise the path traversed by each flow are listed in Table 2.1.

We define the set $\mathcal{E} := [9]$ to be the indices of the edges in the network. The cost of each agent is $f_i(x_i) = -\delta_i \log(1 + x_i)$, and we have selected $\delta_i = 100$ for all $i \in [8]$. The network also has an

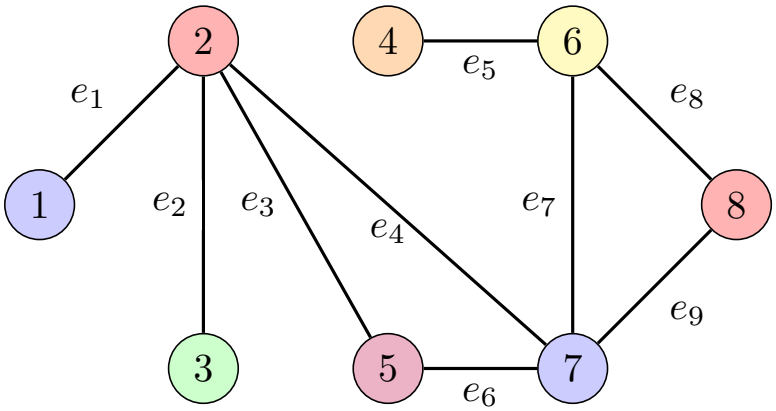

Figure 2.3: The network across which 8 agents route traffic in simulating Algorithm 2.2. There are 9 edges, each with a maximum capacity of 10, and 8 nodes. The edges used by each agent are listed in Table 2.1.

Table 2.1: The edges traversed by each agent's flow in simulating Algorithm 2.2.

Agent Number	Start Node$\rightarrow$End Node	Edges Traversed
1	$1 \rightarrow 7$	e_1, e_3, e_6
2	$2 \rightarrow 8$	e_4, e_7, e_8
3	$3 \rightarrow 4$	e_2, e_4, e_7, e_5
4	$5 \rightarrow 6$	e_3, e_4, e_7
5	$1 \rightarrow 4$	e_1, e_3, e_6, e_7, e_5
6	$3 \rightarrow 8$	e_2, e_4, e_9
7	$4 \rightarrow 5$	e_5, e_8, e_9, e_6
8	$6 \rightarrow 2$	e_7, e_4

associated congestion cost $c(x) = \frac{1}{20}x^T A^T Ax$, where

$$A_{k,i} = \begin{cases} 1 & \text{if flow } i \text{ traverses edge } k \\ 0 & \text{otherwise} \end{cases}$$

defines the network's adjacency matrix A.

Each edge in the network is subject to capacity constraints, expressed by requiring $Ax \leq b$, where $b_i = 10$ for all $i \in \mathcal{E}$. In addition, each flow rate is confined to $[0, 10]$, giving $X = [0, 10]^8$. To demonstrate the effects of different values of α and β, three simulations were run: the first with $\alpha = \beta = 0.1$, the second with $\alpha = \beta = 0.01$, and the third with $\alpha = \beta = 0.001$. By sweeping α and β across three orders of magnitude, we demonstrate the speed-accuracy tradeoff discussed in Remark 2.3. For each α, we take $\gamma = 2/(L_p + \alpha)$, and we take $\rho = 0.9\rho_0$ for each (α, β) pair.

2.6.2 Implementation and Numerical Results

The implementation of the above problem allowed as many quantities as possible to be random to demonstrate asynchronous behavior. The time between cloud updates was a random integer chosen from the range 5 to 100 (inclusive) with uniform probability, and this number represents the number of ticks of the virtual clock K between k_t and k_{t+1}. At each tick of K, each agent computed a state update with probability $p_{update} = 0.05$ for all agents.

The communication graph at each tick of K was an Erdős-Rényi graph [37, Chapter 5], which is a random graph wherein each edge appears with some probability independently of all other edges. We chose $p_{edge} = 0.05$, so that at each time $k \in K$ we had the graph $G(k) = (V, E(k))$, where $\mathbb{P}[(i, j) \in E(k)] = 0.05$ for all i and j in each other's essential neighborhoods. The communication graph in this case was undirected so that $(i, j) \in E(k)$ means that agent i sends its state to agent j at time k, and vice versa. All transmissions are received instantaneously. The times at which the agents sent their states to the cloud were chosen to be randomly generated times between k_t and k_{t+1} which were uniformly distributed and independent of all communications and

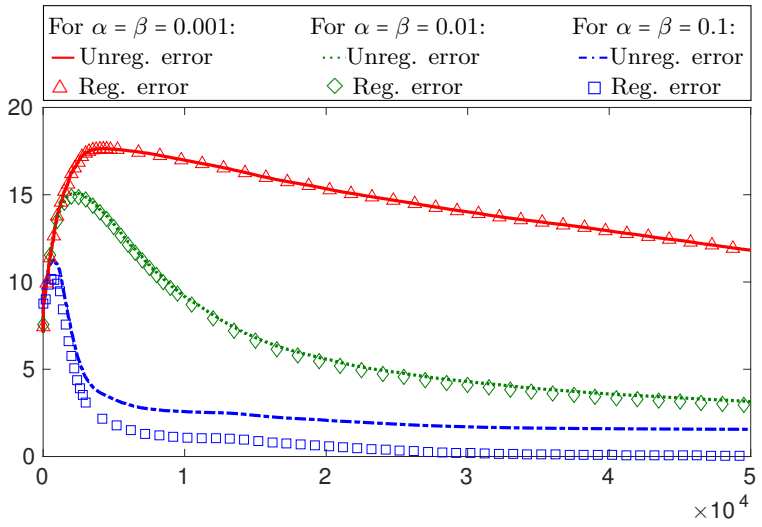

Figure 2.4: The values of the unregularized primal error, $\|x_t^c - \hat{x}\|$ (lines), and regularized primal error, $\|x_t^c - \hat{x}_\kappa\|$ (shapes), for the simulation runs of Algorithm 2.2 using $\alpha = \beta = 0.001$ (top pair of curves), $\alpha = \beta = 0.01$ (middle pair of curves), and $\alpha = \beta = 0.1$ (bottom pair of curves). It is evident that larger regularization parameters lead to faster decreases in error, indicating faster convergence.

Table 2.2: The final primal errors in each simulation of Algorithm 2.2. As predicted by Remark 2.3, smaller regularization parameters do indeed lead to smaller errors.

Value of α and β	Final reg. error $\|x_t^c - \hat{x}_\kappa\|$	Final unreg. error $\|x_t^c - \hat{x}\|$	Max final value of g_j
0.1	$1.352 \cdot 10^{-12}$	8.616	1.948
0.01	$7.129 \cdot 10^{-13}$	0.223	0.252
0.001	$1.414 \cdot 10^{-11}$	0.0237	0.0262

computations.

Each of the three simulation runs was run until it converged. In Figure 2.4 we see three pairs of curves: the uppermost pair corresponds to $\alpha = \beta = 0.001$, the middle pair corresponds to $\alpha = \beta = 0.01$, and the lowest pair corresponds to $\alpha = \beta = 0.1$. Each pair plots the unregularized primal error $\|x_t^c - \hat{x}\|$ for each run using lines, and the regularized primal error $\|x_t^c - \hat{x}_\kappa\|$ is plotted using shapes. Figure 2.5 similarly shows the regularized and unregularized dual errors, $\|\mu(t) - \hat{\mu}\|$ and $\|\mu(t) - \hat{\mu}_\kappa\|$, using lines and shapes, respectively, for each choice of regularization parameters.

Figure 2.4 shows that all error curves initially increase, following which they decrease at different rates, with larger regularization parameters clearly leading to faster decreases in error. The

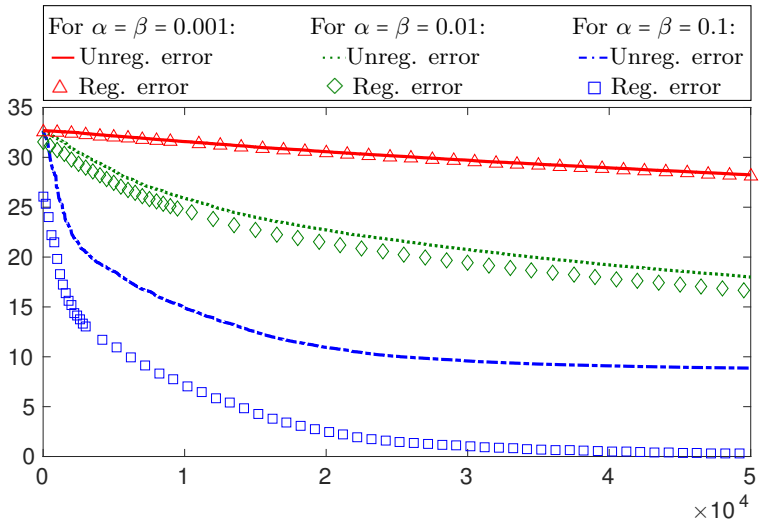

Figure 2.5: The values of the unregularized dual error, $\|\mu(t) - \hat{\mu}\|$ (lines), and the regularized dual error, $\|\mu(t) - \hat{\mu}_\kappa\|$ (shapes), for the simulation runs of Algorithm 2.2 using $\alpha = \beta = 0.001$ (top pair of curves), $\alpha = \beta = 0.01$ (middle pair of curves), and $\alpha = \beta = 0.1$ (bottom pair of curves). As in Figure 2.4, we see that increasing the regularization parameters α and β results in faster convergence to a final value.

Table 2.3: The final dual errors in each simulation of Algorithm 2.2, which show that increasing regularization parameters does indeed result in larger errors.

Value of α and β	Final reg. error $\|\mu(t) - \hat{\mu}_\kappa\|$	Final unreg. error $\|\mu(t) - \hat{\mu}\|$
0.1	$7.507 \cdot 10^{-12}$	8.616
0.01	$4.600 \cdot 10^{-12}$	1.573
0.001	$1.056 \cdot 10^{-10}$	0.174

final primal errors for each simulation run are given in Table 2.2, where we see that all three runs numerically converge almost exactly to $\hat{x}_\kappa$. We also see that smaller regularization parameters decrease final primal errors, as predicted by Remark 2.3.

Figure 2.5 shows behavior in the dual space similar to that shown in Figure 2.4. All curves appear to decrease monotonically, with larger values of α and β clearly showing a faster rate of decrease. And as with the primal space, one finds that larger regularization parameters lead to larger errors in the dual space; final dual error values are shown in Table 2.3 wherein one finds that all three runs virtually exactly reach $\hat{\mu}_\kappa$ and, indeed, decreasing α and β decreases the final unregularized dual error.

We see in Figures 2.4 and 2.5 that increasing the regularization parameters leads to faster convergence, and this same phenomenon was observed numerically in [33]. However, a key numerical difference between our results and some of those in earlier works, e.g., [26] and [27], is the initial increase in distance to the optimum seen in Figure 2.4. This increase is unavoidable due to the agents sharing information asynchronously and is typical in simulation runs of Algorithm 2.

2.7 Conclusion

In this chapter, an asynchronous multi-agent optimization algorithm for constrained problems was presented. It was shown that the dual variable must be kept synchronized across the agents, though their primal updates can occur independently and with arbitrary timing. The method presented used a Tikhonov regularization and a multi-agent gradient projection method to approximately find saddle points of the regularized Lagrangian asynchronously. As a result, we find that asynchronous optimization is not only possible, but also that it can lead to accurate, useful results, despite disagreements about the value of data across the network.

An advantage of this work is that it develops an optimization algorithm which is *descriptive* rather than *prescriptive*, i.e., it is able to provide convergence rate estimates for any communications in the system, rather than specifying a mandatory algorithm and the associated performance guarantees of that algorithm. This flexibility is particularly well-suited to applications in which communications are unreliable, such as in a team of agents spread very far apart, or in a convoy in a hostile environment in which communications are being jammed by an adversary.

By stating convergence results in terms of $c(t)$ without specifying what $c(t)$ must be, one can also answer questions about which edges would be most useful to activate at any point in time; if any collection of edges would serve to complete an additional cycle, the corresponding communications can be prioritized over others in order to help convergence. In this fashion, the theoretical results of this chapter can be used to inform network behavior to aid in successful use of this work. The next chapter is entirely devoted to determining how often cycles can be expected to occur in a general network coordination setting.

CHAPTER 3

CONNECTIVITY AND COMPLETENESS OF UNIONS OF RANDOM GRAPHS

In Chapter 2 it was shown that the rate of convergence of Algorithm 2.2 is heavily dependent upon the frequency with which required communications are carried out (cf. Theorems 2.1 and 2.2). In particular, it was shown that the number of communication cycles completed over time, as defined in Definition 2.4, appears in Theorems 2.1 and 2.2, and these theorems quantify the extent to which communication affects convergence. Accordingly, one may wonder how many cycles can be expected to occur in any given period of time.

Toward answering this question, we explore the rate at which cycles are completed for multi-agent communications modeled by random graphs. Random graphs were used in the simulations presented in Section 2.6 and can model a variety of behaviors in multi-agent systems. In some cases, the motivation for representing a communication network using random graphs comes from agents using an interaction protocol that is randomized by design, such as in a gossip-like algorithm [25]. In other cases, unreliable communications due to poor channel quality, interference, and other factors can be effectively represented by a random communication graph [37], and the work here applies to each of these scenarios. Random graphs are also commonly used to model asynchronous communications, such as those studied in Chapter 2, and this work can be directly used in the framework developed there.

Motivated by work in multi-agent systems, we consider networks of a fixed size, and we examine random graphs generated by the Erdős-Rényi model[1] [38], in which each possible edge in a graph is present with a fixed probability and is independent of all other edges. The Erdős-Rényi model is used because it accurately captures the behavior of many multi-agent systems, including

[1] Throughout this chapter, the phrase "random graphs" always refers to Erdős-Rényi graphs.

the aforementioned cases of networks with intermittent and unreliable communications [39], and the behavior of some variants of synchronous gossip algorithms [25]. Our approach consists of calculating the first four (matrix-valued) moments of the Laplacian associated with a finite union of random graphs, and then computing the eigenvalues of these matrices. These eigenvalues are then used to bound the expectation and variance of the algebraic connectivity [40] of this union of graphs. By constructing these bounds in terms of a network's size and edge probability, it is shown that a union of Erdős-Rényi graphs can attain some specified expected algebraic connectivity, provided the number of graphs in the union exceeds a threshold which we compute. We also lower-bound the probability with which the algebraic connectivity of a union of random graphs exceeds some specified value.

To provide worst-case bounds for the amount of time needed to complete a desired number of communication cycles, we focus in part on complete graphs. The greatest number of communication links needed in the framework of Chapter 2 comes from cases in which communications are all-to-all, and thus form a complete graph over time. For problems in which communications are not all-to-all, the communications required comprise a subset of the edges in a complete graph, and thus communications cycles will be completed in those cases before they are completed in the all-to-all case. However, the value of examining complete unions of Erdős-Rényi graphs is that the results obtained from this analysis provide a worst-case bound, over all possible problem formulations, for the time required to complete a specified number of cycles. This worst-case bound holds precisely because the complete graph contains all other possible communication graphs. Therefore, we study the number of graphs required for their union to be approximately complete (in a sense to be defined below). As an auxiliary result, we are also able to determine the number of graphs needed in a union before that union is expected to form a connected graph; this result may be of independent interest as a number of works rely on communication graphs having connected unions over finite intervals (or a related variant of this assumption), including [24], [41]–[55].

Both results rely in some form on computing eigenvalues of random matrices, and there is an established literature dedicated to doing so [56], [57], including for eigenvalues of random

symmetric matrices [58]–[60], and eigenvalues of random graphs' Laplacians specifically [61]. A common approach to estimating or computing the eigenvalues of a random symmetric matrix is to let the size of the matrix get arbitrarily large [56], [59], [60], [62]. In graph theory, this approach corresponds to letting the number of nodes in a graph grow arbitrarily large, and it has seen use in spectral graph theory because it allows one to rigorously state results that hold for almost all graphs [63].

In the study of multi-agent systems, one is often interested in networks of a fixed, small size, such as in [64]–[66], and this makes results for asymptotically large networks less applicable in some cases. As a result, we derive convergence results in terms of a network's size without taking it to grow asymptotically large. In addition, there are a number of graph theoretic results that estimate eigenvalues of random graphs' Laplacians when edge probabilities bear some known relationship to the size of the network, e.g., [67], [68]. In cases where a random graph is used to model unreliable communications, there is no guarantee that such a relationship will hold as the quality of communication channels can depend upon a wide variety of external factors. Therefore, we allow edge probabilities to take values independent of the network's size, and we state our results in terms of both a network's size and its edge probability, without making assumptions about either or about a relationship between them.

The rest of this chapter is organized as follows. Section 3.1 reviews the necessary elements of graph theory, including random graphs, and provides formal statements of the problems we solve. Then, Section 3.2 computes the first four moments of a random graph's Laplacian. Next, Section 3.3 presents the main results of the chapter in the form of bounds on the number of graphs required in a union to attain a specified bound on the union's algebraic connectivity. Then, Section 3.4 provides numerical results with various graph parameters. Finally, Section 3.5 concludes the chapter.

3.1 Review of Graph Theory

In this section, we review the basic elements of graph theory required for the remainder of the chapter. We first introduce unweighted, undirected graphs and then review random graphs. Then we formally state the problems under consideration.

3.1.1 Basic Graph Theory

All graphs in this chapter are assumed to be simple (i.e., no self loops), unweighted, and undirected. Such graphs are defined by pairwise relationships over a finite set of nodes or vertices. Suppose that a graph has a set V of n vertices, with $n \in \mathbb{N}$, and index these vertices over the set $\{1, \dots, n\}$. We define the *edge set*

$$E \subseteq V \times V,$$

and say there is an edge between nodes i and j if $(i, j) \in E$. A graph G is then formally defined as the 2-tuple $G = (V, E)$. Throughout this chapter, all edges are undirected and an edge $(i, j) \in E$ is not distinguished from the edge $(j, i) \in E$. We do not allow self loops and therefore $(i, i) \notin E$ for all i and all graphs G. One main focus of this chapter is on *connected* graphs, which we define now.

Definition 3.1 *(E.g., [69]) A graph G is called* connected *if, for all $i \in [n]$ and $j \in [n]$, $i \neq j$, there is a sequence of edges one can traverse from node i to node j, i.e., there is a sequence of indices $\{i_\ell\}_{\ell=1}^{k}$ and nodes $\{v_p\}_{p=i_1}^{i_k}$ such that E contains all of the edges*

$$(i, v_{i_1}),\ (v_{i_1}, v_{i_2}),\ (v_{i_2}, v_{i_3}),\ \dots,\ (v_{i_{k-1}}, v_{i_k}),\ (v_{i_k}, j).$$

$\triangle$

The results of this chapter are stated in terms of graph Laplacians, which depend upon the degree and adjacency matrices associated with a graph. The degree of node i is defined as the total

number of edges that connect node i to some other node. Using $|\cdot|$ to denote the cardinality of a set and using d_i to denote the degree of node i, we have

$$d_i = \big|\{j \mid (i,j) \in E\}\big|.$$

The $n \times n$ degree matrix associated with a graph G is then defined as

$$D(G) = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_n \end{pmatrix},$$

which will be written simply as D when the graph G is understood.

The $n \times n$ adjacency matrix associated with G, denoted $A(G)$, is defined element-wise as

$$a_{i,j} = \begin{cases} 1 & (i,j) \in E \\ 0 & \text{otherwise,} \end{cases}$$

where $a_{i,j}$ is the $i^{th}j^{th}$ entry in $A(G)$. Note that, because $(i,j) \in E$ implies $(j,i) \in E$, $A(G)$ is a symmetric matrix. In addition, the absence of self loops results in $A(G)$ having zeroes on its main diagonal for all graphs G. We will simply write A when G is clear from context.

The Laplacian of a graph G is then defined as

$$L(G) = D(G) - A(G),$$

which will be written simply as L when G is unambiguous. The results of this chapter rely in particular on spectral properties of L, which is symmetric and positive-definite for any undirected, unweighted graph [69]. Letting $\lambda_k(\cdot)$ denote the k^{th} smallest eigenvalue of a matrix, it is known

that $\lambda_1(L) = 0$ for all graph Laplacians [37], and thus we have

$$0 = \lambda_1(L) \leq \lambda_2(L) \leq \cdots \leq \lambda_n(L). \tag{3.1}$$

The value of $\lambda_2(L)$ is central to the work in this chapter and some other works in graph theory, and it gives rise to the following definition.

Definition 3.2 *(From [40]) The* algebraic connectivity *of a graph G is the second smallest eigenvalue of its Laplacian, $\lambda_2(L)$, and G is connected if and only if $\lambda_2(L) > 0$.* $\triangle$

This chapter is dedicated to studying the statistical properties of λ_2 for unions of random graphs. Toward doing so, we now review the necessary elements of the theory of random graphs.

3.1.2 Random Graphs

A common model for random graphs is the Erdős-Rényi model, originally published in [38], and we use it here because it accurately captures the behavior of two cases of interest. First, some algorithms are randomized by design, such as gossip algorithms [25], and Erdős-Rényi graphs can model the behavior of such algorithms in some cases. Second, members of a network sometimes share information over communication channels which are intermittently lost and regained, and this behavior is well-modeled by Erdős-Rényi graphs as well [39]. This model takes two parameters to generate random graphs: a number of nodes $n \in \mathbb{N}$ and an edge probability[2] $p \in (0, 1)$. The Erdős-Rényi model generates graphs on n nodes whose edge sets contain each possible edge with probability p, independent of all other edges. Formally, for each admissible i and j, we have

$$\mathbb{P}[(i, j) \in E] = p \text{ and } \mathbb{P}[(i, j) \notin E] = 1 - p.$$

[2]The cases $p = 0$ and $p = 1$ provide edgeless graphs and complete graphs, respectively, and are omitted because their behavior is deterministic.

An alternative characterization that we use later can be stated in terms of the elements of the adjacency matrix of a random graph: for n nodes and edge probability p, we find

$$\mathbb{P}[a_{i,j} = 1] = \mathbb{P}[a_{j,i} = 1] = p \text{ and } \mathbb{P}[a_{i,j} = 0] = \mathbb{P}[a_{j,i} = 0] = 1 - p.$$

Thus each $a_{i,j}$ is a Bernoulli random variable.

We use $\mathcal{G}(n, p)$ to denote the sample space of all possible random graphs generated by the Erdős-Rényi model on n nodes with edge probability p, and we use $\mathcal{L}(n, p)$ to denote the set of Laplacians of all such graphs. We also use the notation $q := 1 - p$. In the study of multi-agent systems, it is also common for some algorithms and results to be stated in terms of union graphs, which we define now.

Definition 3.3 *For a collection of graphs $\{G_k = (V, E_k)\}_{k=1}^N$ defined on the same node set V, the union of these graphs, denoted U_N, is defined as*

$$U_N := \bigcup_{k=1}^{N} G_k = \left(V, \cup_{k=1}^N E_k\right),$$

i.e., the union graph *U_N contains all edges in all N graphs that comprise the union.* △

With these basic concepts established, we now formally state the problems that are the focus of this chapter. We begin with a problem statement concerning completeness of unions of random graphs.

Problem 3.1 *Fix $n \in \mathbb{N}$ and $p \in (0, 1)$, and let λ_{comp} denote the algebraic connectivity of a complete graph. Find $N_{min} \in \mathbb{N}$ such that $\mathbb{E}[\lambda_2(U_N)] \approx \lambda_{comp}$ for all $N \geq N_{min}$, and lower-bound $\mathbb{P}[\lambda_2(U_N) \geq \lambda_{comp}]$ for all $N \geq N_{min}$, where U_N is given by*

$$U_N = \bigcup_{k=1}^{N} G_k,$$

and $G_k \in \mathcal{G}(n, p)$ for all $k \in \{1, \ldots, N\}$. ♦

The second problem statement concerns connected unions of random graphs.

Problem 3.2 *Fix $n \in \mathbb{N}$ and $p \in (0,1)$, and let λ_{min} denote the minimum algebraic connectivity among all connected graphs. Find $N_{min} \in \mathbb{N}$ such that $\mathbb{E}[\lambda_2(U_N)] \geq \lambda_{min}$ for all $N \geq N_{min}$, and lower-bound $\mathbb{P}[\lambda_2(U_N) \geq \lambda_{min}]$ for all $N \geq N_{min}$, where U_N is defined as it is in Problem 3.1.* ♦

One approach to analyzing random graphs that has seen use in the graph theory literature consists of taking the limit as n goes to infinity, with the benefit of this approach being the ability to rigorously determine which properties hold for almost all graphs. However, motivated by the study of multi-agent systems, we are interested in networks of fixed size and therefore develop our results in terms of a fixed value of n. Toward doing so, we keep n fixed (but unspecified), and we compute the first four moments of a random graph's Laplacian in the next section.

3.2 Computing Moments of the Laplacian of a Random Graph

This section provides several technical lemmas that will be used in the next section to derive the main results of this chapter. To enable those results, it will be necessary to have the first four moments of the Laplacian of a random graph, which we compute in this section.

First, we have the following lemma concerning eigenvalues of a matrix of the form $aI + b(J - I)$, where I is the $n \times n$ identity matrix, and J is the matrix of ones of size $n \times n$.

Lemma 3.1 *A matrix M of the form $aI + b(J - I)$, namely*

$$M = \begin{pmatrix} a & b & \cdots & b \\ b & a & \cdots & b \\ \vdots & \vdots & \ddots & \vdots \\ b & b & \cdots & a \end{pmatrix} \in \mathbb{R}^{n \times n},$$

has $a + (n-1)b$ as an eigenvalue with multiplicity one and $a - b$ as an eigenvalue with multiplicity $n - 1$.

Proof: We proceed using a series of row operations that preserves the characteristic polynomial of M. We see that

$$|M - \lambda I| = \left| \begin{pmatrix} a-\lambda & b & \cdots & b \\ b & a-\lambda & \cdots & b \\ \vdots & \vdots & \ddots & \vdots \\ b & b & \cdots & a-\lambda \end{pmatrix} \right|.$$

Next, we add rows 2 through n to row 1, giving

$$|M - \lambda I| = \big(a + (n-1)b - \lambda\big) \left| \begin{pmatrix} 1 & 1 & \cdots & 1 \\ b & a-\lambda & \cdots & b \\ \vdots & \vdots & \ddots & \vdots \\ b & b & \cdots & a-\lambda \end{pmatrix} \right|.$$

Subtracting b times row 1 from each other row, we find

$$|M - \lambda I| = \big(a + (n-1)b - \lambda\big) \left| \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 0 & a-b-\lambda & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a-b-\lambda \end{pmatrix} \right|,$$

where the matrix on the right-hand side is upper-triangular. The determinant on the right-hand side is then the product of the diagonal entries of that matrix, resulting in

$$|M - \lambda I| = (a + (n-1)b - \lambda)(a - b - \lambda)^{n-1},$$

whose roots are indeed $a + (n-1)b$, with multiplicity 1, and $a - b$ with multiplicity $n - 1$. ■

We now derive the expected value of the first four powers of a random graph's Laplacian.

Lemma 3.2 *Let a number of nodes $n \in \mathbb{N}$ and edge probability $p \in (0, 1)$ be given. The Laplacian*

L of a graph $G \in \mathcal{G}(n,p)$ satisfies

$$\begin{aligned}
\mathbb{E}[L] &= p(nI - J) \\
\mathbb{E}[L^2] &= \left[(n-2)p^2 + 2p\right](nI - J) \\
\mathbb{E}[L^3] &= \left[(n-2)(n-4)p^3 + 6(n-2)p^2 + 4p\right](nI - J) \\
\mathbb{E}[L^4] &= \left[(n-7)(n-3)(n-2)p^4 + 6(2n-7)(n-2)p^3 + 25(n-2)p^2 + 8p\right](nI - J).
\end{aligned}$$

Proof sketch: The general form of graph Laplacian for $G \in \mathcal{G}(n,p)$ is

$$\begin{pmatrix}
\sum_{\substack{j=1 \\ j\neq 1}}^{n} a_{1,j} & -a_{1,2} & \cdots & -a_{1,n} \\
-a_{1,2} & \sum_{\substack{j=1 \\ j\neq 2}}^{n} a_{2,j} & \cdots & -a_{2,n} \\
\vdots & \vdots & \ddots & \vdots \\
-a_{1,n} & -a_{2,n} & \cdots & \sum_{\substack{j=1 \\ j\neq n}}^{n} a_{n,j}
\end{pmatrix},$$

where each term $a_{i,j}$ is a Bernoulli random variable with expectation equal to p. The off-diagonal entries of L have $\mathbb{E}[L_{ij}] = \mathbb{E}[-a_{i,j}] = -p$, while linearity of $\mathbb{E}[\cdot]$ gives $\mathbb{E}[L_{ii}] = (n-1)p$ for diagonal entries of L. From this we find

$$\begin{aligned}
\mathbb{E}[L] &= \begin{pmatrix}
(n-1)p & 0 & \cdots & 0 \\
0 & (n-1)p & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & (n-1)p
\end{pmatrix} - \begin{pmatrix}
0 & p & \cdots & p \\
p & 0 & \cdots & p \\
\vdots & \vdots & \ddots & \vdots \\
p & p & \cdots & p
\end{pmatrix} \\
&= (n-1)pI - p(J - I) \\
&= p(nI - J).
\end{aligned}$$

Computing the expectation of the entries of L^2 requires one to consider two cases. The diagonal entries of L^2 are formed by the product of row i of L with column i of L (and by symmetry of L these are identical), while the off-diagonal entries result from the product of row i and column j

of L (which are not identical when $i \neq j$). It is important to note that $a_{i,j}^2 = a_{i,j}$ because $a_{i,j}$ is a Bernoulli random variable. As a result, when computing expectations one finds that $\mathbb{E}[a_{i,j}^k] = p$ for all $k \in \mathbb{N}$, while products of ℓ distinct off-diagonal entries of A have expectation equal to p^ℓ due to all $a_{i,j}$ terms being mutually independent.

In the case of $\mathbb{E}[L^2]$, a diagonal entry takes the form

$$\begin{aligned}\mathbb{E}\left[(L^2)_{ii}\right] &= \mathbb{E}\left[\sum_{\substack{p=1\\p\neq i}}^{n} a_{i,p}^2 + \left(\sum_{\substack{q=1\\q\neq i}}^{n} a_{i,q}\right)^2\right] \\ &= (n-1)(n-2)p^2 + 2(n-1)p,\end{aligned}$$

while an off-diagonal entry takes the form

$$\begin{aligned}\mathbb{E}[(L^2)_{ij}] &= \mathbb{E}\left[\sum_{\substack{k=1\\k\neq i\\k\neq j}}^{n} a_{i,k}a_{k,j}\right] - \mathbb{E}\left[a_{i,j}\sum_{\substack{p=1\\p\neq i}}^{n} a_{i,p}\right] - \mathbb{E}\left[a_{i,j}\sum_{\substack{q=1\\q\neq i}}^{n} a_{q,j}\right] \\ &= -(n-2)p^2 - 2p.\end{aligned}$$

Then we find

$$\begin{aligned}\mathbb{E}[L^2] &= \big((n-1)(n-2)p^2 + 2(n-1)p\big)I + \big(-(n-2)p^2 - 2p\big)(J-I) \\ &= \big((n-2)p^2 + 2p\big)(nI - J),\end{aligned}$$

as above.

Iterating this process, one finds that a diagonal entry of L^3 takes the form

$$\begin{aligned}(L^3)_{ii} &= \left[\left(\sum_{\substack{j=1\\j\neq i}}^{n} a_{ij}\right)^2 + \sum_{\substack{j=1\\j\neq i}}^{n} a_{ij}^2\right]^2 + \sum_{\substack{j=1\\j\neq i}}^{n}\left(-a_{ij}\left[\sum_{\substack{k=1\\k\neq i}}^{n} a_{ik} + \sum_{\substack{k=1\\k\neq j}}^{n} a_{jk}\right] + \sum_{\substack{\ell=1\\\ell\neq i\\\ell\neq j}}^{n} a_{i\ell}a_{\ell j}\right)^2 \\ &= \left(\sum_{\substack{j=1\\j\neq i}}^{n} a_{ij}\right)^4 + 2\left(\sum_{\substack{j=1\\j\neq i}}^{n} a_{ij}\right)^2 \cdot \sum_{\substack{j=1\\j\neq i}}^{n} a_{ij}^2 + \left(\sum_{\substack{j=1\\j\neq i}}^{n} a_{ij}^2\right)^2 \\ &\quad + \sum_{\substack{j=1\\j\neq i}}^{n}\left(a_{ij}^2\left[\sum_{\substack{k=1\\k\neq i}}^{n} a_{ik} + \sum_{\substack{k=1\\k\neq j}}^{n} a_{jk}\right]^2 - 2a_{ij}\left[\sum_{\substack{k=1\\k\neq i}}^{n} a_{ik} + \sum_{\substack{k=1\\k\neq j}}^{n} a_{jk}\right] + \left(\sum_{\substack{\ell=1\\\ell\neq i\\\ell\neq j}}^{n} a_{i\ell}a_{\ell j}\right)^2\right),\end{aligned}$$

while an off-diagonal entry has the general form

$$\begin{aligned}(L^3)_{ij} &= \sum_{\substack{r=1\\r\neq i\\r\neq j}}^{n} a_{rj}\left(\sum_{\substack{k=1\\k\neq i\\k\neq r}}^{n} a_{i,k}a_{k,r} - a_{i,r}\sum_{\substack{p=1\\p\neq i}}^{n} a_{i,p} - a_{i,r}\sum_{\substack{q=1\\q\neq i}}^{n} a_{q,r}\right) + a_{ij}\left(\sum_{\substack{p=1\\p\neq i}}^{n} a_{i,p}^2 + \left(\sum_{\substack{q=1\\q\neq i}}^{n} a_{i,q}\right)^2\right) \\ &\quad - \left(\sum_{\substack{k=1\\k\neq j}}^{n} a_{j,k}\right)\left(\sum_{\substack{k=1\\k\neq i\\k\neq j}}^{n} a_{i,k}a_{k,j} - a_{i,j}\sum_{\substack{p=1\\p\neq i}}^{n} a_{i,p} - a_{i,j}\sum_{\substack{q=1\\q\neq i}}^{n} a_{q,j}\right)\end{aligned}$$

and computing the expected value of these terms gives the expected value of L^3.

Continuing this process, for L^4 we find

$$(L^4)_{ii} = \left\{ \left[\left(\sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij} \right)^2 + \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}^2 \right]^2 + \sum_{\substack{j=1 \\ j\neq i}}^{n} \left(-a_{ij} \left[\sum_{\substack{k=1 \\ k\neq i}}^{n} a_{ik} + \sum_{\substack{k=1 \\ k\neq j}}^{n} a_{jk} \right] + \sum_{\substack{\ell=1 \\ \ell\neq i \\ \ell\neq j}}^{n} a_{i\ell} a_{\ell j} \right)^2 \right\}^2$$
$$+ \sum_{\substack{j=1 \\ j\neq i}}^{n} \left\{ \sum_{\substack{r=1 \\ r\neq i \\ r\neq j}}^{n} a_{rj} \left(\sum_{\substack{k=1 \\ k\neq i \\ k\neq r}}^{n} a_{i,k} a_{k,r} - a_{i,r} \sum_{\substack{p=1 \\ p\neq i}}^{n} a_{i,p} - a_{i,r} \sum_{\substack{q=1 \\ q\neq i}}^{n} a_{q,r} \right) + a_{ij} \left(\sum_{\substack{p=1 \\ p\neq i}}^{n} a_{i,p}^2 + \left(\sum_{\substack{q=1 \\ q\neq i}}^{n} a_{i,q} \right)^2 \right) \right.$$
$$\left. - \left(\sum_{\substack{k=1 \\ k\neq j}}^{n} a_{j,k} \right) \left(\sum_{\substack{k=1 \\ k\neq i \\ k\neq j}}^{n} a_{i,k} a_{k,j} - a_{i,j} \sum_{\substack{p=1 \\ p\neq i}}^{n} a_{i,p} - a_{i,j} \sum_{\substack{q=1 \\ q\neq i}}^{n} a_{q,j} \right) \right\}^2$$

for the general form of a diagonal entry. By squaring the general form of L^2, we find

$$(L^4)_{ij} = \sum_{\substack{t=1 \\ t\neq i \\ t\neq j}}^{n} \left\{ \left(\sum_{\substack{k=1 \\ k\neq i \\ k\neq t}}^{n} a_{i,k} a_{k,t} - a_{i,t} \sum_{\substack{p=1 \\ p\neq i}}^{n} a_{i,p} - a_{i,t} \sum_{\substack{q=1 \\ q\neq i}}^{n} a_{q,t} \right) \left(\sum_{\substack{k=1 \\ k\neq i \\ k\neq t}}^{n} a_{t,k} a_{k,i} - a_{t,i} \sum_{\substack{p=1 \\ p\neq t}}^{n} a_{t,p} - a_{t,i} \sum_{\substack{q=1 \\ q\neq t}}^{n} a_{q,i} \right) \right\}$$
$$+ \left(\sum_{\substack{p=1 \\ p\neq i}}^{n} a_{i,p}^2 + \left(\sum_{\substack{q=1 \\ q\neq i}}^{n} a_{i,q} \right)^2 \right) \left(\sum_{\substack{k=1 \\ k\neq i \\ k\neq j}}^{n} a_{i,k} a_{k,j} - a_{i,j} \sum_{\substack{p=1 \\ p\neq i}}^{n} a_{i,p} - a_{i,j} \sum_{\substack{q=1 \\ q\neq i}}^{n} a_{q,j} \right)$$
$$+ \left(\sum_{\substack{p=1 \\ p\neq j}}^{n} a_{j,p}^2 + \left(\sum_{\substack{q=1 \\ q\neq j}}^{n} a_{j,q} \right)^2 \right) \left(\sum_{\substack{k=1 \\ k\neq i \\ k\neq j}}^{n} a_{i,k} a_{k,j} - a_{i,j} \sum_{\substack{p=1 \\ p\neq i}}^{n} a_{i,p} - a_{i,j} \sum_{\substack{q=1 \\ q\neq i}}^{n} a_{q,j} \right)$$

for an off-diagonal entry of L^4. Computing the expectation of these two terms then gives the expected form of L^4 as well. ■

These four moments of L will be used directly in solving Problems 3.1 and 3.2 in the next

section. As stated, these moments of L apply to single random graphs. However, under the Erdős-Rényi model, a union of random graphs can itself be treated as a single random graph with a different edge probability, allowing these results to be easily extended to unions of random graphs. This is formally stated and shown in the next lemma.

Lemma 3.3 *Let $\mathcal{G}(n,p)$ denote the sample space of all Erdős-Rényi graphs on n vertices with edge probability p, and let $\mathcal{U}_N(n,p)$ denote the space of unions of N graphs from $\mathcal{G}(n,p)$. Then*

$$\mathcal{U}_N(n,p) = \mathcal{G}(n, 1-(1-p)^N),$$

i.e., a union of Erdős-Rényi graphs is itself an Erdős-Rényi graph.

Proof: Consider some $G \in \mathcal{U}_N(n,p)$. Fix any admissible node indices i and j. Then an edge is absent between i and j only if it is absent in all N graphs that comprise G. That is, an edge between i and j is absent in G with probability q^N. Then that edge is present with probability $1-q^N = 1-(1-p)^N$. ■

Thus we may develop our results for single Erdős-Rényi graphs with the knowledge that these results apply to unions of such graphs under a suitable change in edge probability. The final lemma required lets us study the expected eigenvalues of a graph Laplacian (and its powers) by studying the eigenvalues of the expected graph Laplacian. The expected graph Laplacian takes a simple form and its eigenvalues are straightforward to compute, and this lemma will be used to simplify the process of computing moments of the eigenvalues of random graphs' Laplacians. We state the lemma below.

Lemma 3.4 *Let* $\text{eig}(\cdot)$ *denote the set of eigenvalues of a matrix. Then for a graph Laplacian $L \in \mathcal{L}(n,p)$ and $k \in \{1,2,3,4\}$ we have*

$$\mathbb{E}[\text{eig}(L^k)] = \text{eig}(\mathbb{E}[L^k]).$$

Proof: The expected value of any diagonal element of a graph Laplacian takes the form

$$\mathbb{E}[L_{ii}] = \mathbb{E}\left[\sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}\right] = (n-1)p,$$

because the random variables a_{ij} are independent Bernoulli random variables which take value 1 with probability p. Summing these, we find the expected trace of L to be

$$\mathbb{E}[\text{trace}(L)] = \sum_{i=1}^{n} \mathbb{E}[L_{ii}] = n(n-1)p.$$

Denote the eigenvalues of the Laplacian of a random graph by ℓ_i, $1 \leq i \leq n$. Due to the symmetries in the problem and the fact that all off-diagonal elements of L are i.i.d. random variables, along with the diagonal elements simply being sums of these variables, the non-zero eigenvalues of L are equal in expectation. That is, apart from the guaranteed zero eigenvalue $\ell_1 = 0$ which is true of every graph Laplacian (as noted in Equation (3.1)), all other eigenvalues have equal expectation precisely because all off-diagonal entries have the same form and because all diagonal entries do as well. We then have that

$$\sum_{i=2}^{n} \mathbb{E}[\ell_i] = \mathbb{E}[\text{trace}(L)] = n(n-1)p$$

from above, giving

$$\mathbb{E}[\ell_i] = np \tag{3.2}$$

for all $i \in \{2, \ldots, n\}$.

As for $\text{eig}(\mathbb{E}[L])$, we note that

$$\mathbb{E}[L] = p(nI - J),$$

which by Lemma 3.1 has eigenvalues

$$\ell_1 = 0 \text{ and } \ell_i = np \text{ for } k \in \{2, \ldots, n\}. \tag{3.3}$$

Comparing Equations (3.2) and (3.3), we find that

$$\mathbb{E}[\text{eig}(L)] = \text{eig}(\mathbb{E}[L])$$

as desired.

By the same reasoning, one can repeatedly exploit the symmetries of L and its powers to obtain

$$\mathbb{E}[\text{eig}(L^k)] = \text{eig}(\mathbb{E}[L^k])$$

for any $k \in \{2, 3, 4\}$. ■

The next section exploits this result to solve Problems 3.1 and 3.2. Before doing so, we draw an important distinction between the eigenvalues $\{\ell_i\}_{i=1}^n$ studied in this section and the eigenvalues $\{\lambda_i\}_{i=1}^n$ in Section 3.1. The eigenvalues $\{\ell_i\}_{i=1}^n$ comprise an unordered collection and are simply the eigenvalues of a random graph's Laplacian; as a result, each ℓ_i is itself a random variable. In the setting of random graphs, λ_i is then the i^{th} *order statistic* over these random variables, i.e., the i^{th} smallest value realized by any of the random variables in the collection $\{\ell_i\}_{i=1}^n$.

More concretely, using the convention that $\ell_1 = 0$ is always the zero eigenvalue of a random graph's Laplacian (which is guaranteed to exist by Equation (3.1)), the algebraic connectivity of a random graph is defined as

$$\lambda_2 = \min_{2 \leq i \leq n} \ell_i.$$

This section has characterized each ℓ_i, and Section 3.3 uses these results to characterize λ_2.

3.3 Probabilistic Connectivity and Completeness of Unions of Erdős-Rényi Graphs

In this section, we present the main results of this chapter, and using the computations of moments from Section 3.2 we present solutions to Problems 3.1 and 3.2. The emphasis of this section is on deriving probabilistic bounds on the algebraic connectivity of unions of random graphs. To that end, we first review the bounds on algebraic connectivity that correspond to connected and complete graphs. Then we provide bounds on a union graph's algebraic connectivity in terms of the moments of L derived above. Finally, we use these bounds to estimate the number of graphs needed in a union in order to provide either a connected or complete union graph.

3.3.1 Values of λ_2 for Connectivity and Completeness

It was stated in Section 3.1 that the algebraic connectivity of a graph is the second smallest eigenvalue of its associated Laplacian, denoted λ_2, and that a graph is connected if and only if $\lambda_2 > 0$ [40]. This result reveals a subtle point: using Lemmas 3.1 and 3.2, one can see that a random graph's expected Laplacian has a zero eigenvalue and $n-1$ eigenvalues with positive expected value. However, it is intuitively clear that a graph with very small edge probability should not always be a connected graph, despite this positive expected value. This point is resolved by noting that there is a smallest value of λ_2 which is actually attained by a connected graph, i.e., for a connected graph on n vertices, there is a minimum possible value of its algebraic connectivity, which we state now.

Lemma 3.5 *The minimum algebraic connectivity attained by a connected graph on n nodes is*

$$\lambda_2 = 2\left(1 - \cos\frac{\pi}{n}\right).$$

Proof: This follows from Corollary 3.2 and Table 4.4 in [40]. ■

For the case of a complete graph, we have the following result.

Lemma 3.6 *A complete graph on n vertices has algebraic connectivity*

$$\lambda_2 = n.$$

Proof: See Example 2.13 in [37]. ■

In what follows, we will use the results of Lemma 3.5 and Lemma 3.6 in order to estimate the number of graphs needed in a union to expect connectivity and approximate completeness. Presently, we bound the expectation and variance of the algebraic connectivity of a random graph.

3.3.2 Order Statistics of the Eigenvalues of a Random Graph

Denote the eigenvalues of a random graph's Laplacian by ℓ_i for $i \in \{1, \ldots, n\}$. All graphs have at least one zero eigenvalue [40], and we always denote the guaranteed zero eigenvalue by $\ell_1 = 0$. The remaining eigenvalues are unordered with respect to their indices, and this means that the algebraic connectivity of a random graph is

$$\lambda_2 = \min_{2 \leq i \leq n} \ell_i. \tag{3.4}$$

Each ℓ_i is itself a random variable because it is a function of the Bernoulli random variables that comprise a random graph's Laplacian. Equation (3.4) makes it clear that the behavior of λ_2 will depend upon that of the eigenvalues ℓ_i. To formalize this relationship, we first compute the expectation and variance of each ℓ_i and then bound the same quantities for λ_2. We first have the following lemma.

Lemma 3.7 *For all $i \in \{2, \ldots, n\}$,*

$$\mathbb{E}[\ell_i] = np$$

and

$$\text{Var}[\ell_i] = 2npq$$

where $q := 1 - p$.

Proof: From Lemma 3.4, we see that $\mathbb{E}[\ell_i]$ is equal to any non-zero eigenvalue of $\mathbb{E}[L]$. Using Lemma 3.2, we see that

$$\mathbb{E}[L] = p(nI - J) = (n-1)pI - p(J - I).$$

Using Lemma 3.1, this has two distinct eigenvalues: 0, with multiplicity 1, and np with multiplicity $n-1$. Then for $i \in \{2, \ldots, n\}$ we have $\mathbb{E}[\ell_i] = np$ and the first part of the lemma is complete.

For the second part, we see that

$$\text{Var}[\ell_i] = \mathbb{E}[\ell_i^2] - \mathbb{E}[\ell_i]^2. \tag{3.5}$$

It follows from Lemma 3.4 that $\mathbb{E}[\ell_i^2]$ is equal to a non-zero eigenvalue of $\mathbb{E}[L^2]$. Lemma 3.2 provides that

$$\mathbb{E}[L^2] = \left[(n-2)p^2 + 2p\right](nI - J)$$

and Lemma 3.1 gives that a non-zero eigenvalue of this matrix is equal to $n(n-2)p^2 + 2np$. Returning to Equation (3.5) and using the first part of the lemma, we find that

$$\begin{aligned}\text{Var}[\ell_i] &= n(n-2)p^2 + 2np - (np)^2 \\ &= 2npq\end{aligned}$$

as desired.

■

Having computed the expectation and variance of a general eigenvalue of L, we now turn to computing these same quantities for the algebraic connectivity of L, λ_2. To do so, we have the following lemma relating the expectation and variance of individual random variables to the expectation of order statistics over these random variables.

Lemma 3.8 *Let $X_1, \ldots, X_m$ be random variables (not necessarily independent) with common*

mean μ and common variance σ^2. Then the k^{th} order statistic of this collection, i.e., the k^{th} smallest among them, which we denote $X_{k:m}$, has mean bounded according to

$$\mu - \sigma\sqrt{\frac{m-k}{k}} \leq \mathbb{E}[X_{k:m}] \leq \mu + \sigma\sqrt{\frac{k-1}{m-k+1}}.$$

Proof: See [70], Equation (4). ■

Using this lemma, we have the following theorem on the expectation of λ_2 for a random graph.

Lemma 3.9 *Let $G \in \mathcal{G}(n,p)$. Its algebraic connectivity is has expectation bounded according to*

$$\max\left\{np - \sqrt{2n(n-2)pq}, 0\right\} \leq \mathbb{E}[\lambda_2] \leq np, \tag{3.6}$$

where $q = 1 - p$.

Proof: λ_2 is by definition the minimum of the eigenvalues $\ell_2, \ldots, \ell_n$, each of which has $\mu = np$ and $\sigma = \sqrt{2npq}$ from Lemma 3.7. In the setting of Lemma 3.8, we have $m = n - 1$ and $k = 1$, and the result follows by substituting these values into Lemma 3.8. ■

It is possible that the left-hand side of Equation (3.6) is zero for some values of p. In particular, a straightforward calculation shows that the left-hand side of Equation (3.6) is only positive when

$$p > \frac{\sqrt{n^2 + 16(n-2)^2} - n}{4(n-2)},$$

and for p outside this range, Lemma 3.9 does not provide a lower bound on λ_2 beyond its non-negativity (which can be inferred from the non-negativity of each ℓ_i). However, despite this limitation, Lemma 3.9 will be instrumental in solving Problems 3.1 and 3.2 below. Before doing so, we now bound the variance of λ_2 by following an argument similar to that in Lemma 3.9.

Lemma 3.10 *Let $G \in \mathcal{G}(n,p)$. Its algebraic connectivity has variance bounded according to*

$$\begin{aligned}
\text{Var}[\lambda_2] &\leq n(n-2)p^2 + 2np - \left(\max\{0, np - \sqrt{2n(n-2)pq}\}\right)^2 \\
\text{Var}[\lambda_2] &\geq n(n-2)p^2 + 2np - \sigma[\ell_i^2]\sqrt{n-2} - n^2p^2,
\end{aligned}$$

where

$$\sigma[\ell_i^2] = \left(n(n-7)(n-3)(n-2)p^4 + 6n(2n-7)(n-2)p^3 + 25n(n-2)p^2 + 8np - \left(n(n-2)p^2 + 2np\right)^2\right)^{1/2}.$$

Proof: Toward computing $\text{Var}[\lambda_2]$, we bound $\mathbb{E}[\lambda_2^2]$ as was done for $\mathbb{E}[\lambda_2]$ above. To do so, we first need the value of $\sigma[\ell_i^2]$. The variance of the random variable ℓ_i^2 can be computed as

$$\text{Var}[\ell_i^2] = \mathbb{E}[\ell_i^4] - \left(\mathbb{E}[\ell_i^2]\right)^2,$$

where a straightforward application of Lemmas 3.4 and 3.2 gives

$$\mathbb{E}[\ell_i^4] = n(n-7)(n-3)(n-2)p^4 + 6n(2n-7)(n-2)p^3 + 25n(n-2)p^2 + 8np$$

and

$$\mathbb{E}[\ell_i^2] = n(n-2)p^2 + 2np.$$

One then finds

$$\sigma[\ell_i^2] = \left(n(n-7)(n-3)(n-2)p^4 + 6n(2n-7)(n-2)p^3 + 25n(n-2)p^2 + 8np - (n(n-2)p^2 + 2np)^2\right)^{1/2}.$$

Applying Lemma 3.8 to $\mathbb{E}[\lambda_2^2]$ one finds that, with $m = n-1$ and $k = 1$,

$$\mathbb{E}[\lambda_2^2] \leq n(n-2)p^2 + 2np + \sigma[\ell_i^2]\sqrt{\frac{1-1}{n-1}} = n(n-2)p^2 + 2np \tag{3.7}$$

and

$$\mathbb{E}[\lambda_2^2] \geq n(n-2)p^2 + 2np - \sigma[\ell_i^2]\sqrt{n-2}. \tag{3.8}$$

Using Lemma 3.9 and Equations (3.7) and (3.8), we have

$$\begin{aligned}
\text{Var}[\lambda_2] &\geq n(n-2)p^2 + 2np - \sigma[\ell_i^2]\sqrt{n-2} - n^2p^2 \\
\text{Var}[\lambda_2] &\leq n(n-2)p^2 + 2np - \left(\max\left\{np - \sqrt{2n(n-2)pq}, 0\right\}\right)^2,
\end{aligned}$$

as desired. ■

Below we apply these results to unions of random graphs.

3.3.3 Probabilistic Connectivity and Completeness of Unions of Random Graphs

We now present the main results of the chapter. We first focus on connectivity of unions and then examine approximate completeness which, as discussed above, provides a worst-case analysis for the rate at which cycles occur in Algorithm 2.2 in Chapter 2. We have the following result.

Theorem 3.1 *Consider a union of N random graphs on n vertices with edge probability p. Then the expected algebraic connectivity of this union is bounded below by some $\lambda_{lower} \in (0, n)$, i.e., we have*

$$\mathbb{E}[\lambda_2] \geq \lambda_{lower},$$

when

$$n\big(1-(1-p)^N\big) - \sqrt{2n(n-2)\big(1-(1-p)^N\big)(1-p)^N} \geq \lambda_{lower}.$$

Proof: This follows from Lemma 3.3 and Lemma 3.9. ■

Although Lemma 3.6 shows that one requires $\lambda_2 = n$ for completeness of a graph, Theorem 3.1 only applies to values of λ_{lower} that are less than n. This is done because the expectation of λ_2 can never be equal to n for $p \in (0, 1)$, which we state and show formally in the following corollary.

Corollary 3.1 *No finite union of random graphs on n nodes with edge probability p can have expected algebraic connectivity bounded below by n.*

Proof: For all finite N, there is always some non-zero probability of having a non-complete union graph, and thus the expected value of λ_2 must also be less than n. ■

Corollary 3.1 implies that λ_{lower} must be less than n in Theorem 3.1 to compute a finite lower bound on N. Nonetheless, one can set $\lambda_{lower} = n - \varepsilon$ for any $\varepsilon \in (0, n)$ to derive a lower bound on N which attains an "approximately" complete graph, and this will be done shortly. First, considering the general case, we now compute a value of N which guarantees that the expected value of λ_2 is bounded below by some $\lambda_{lower} \in (0, n)$.

Theorem 3.2 *Consider a union of N random graphs on n vertices with edge probability p. The algebraic connectivity of this union is bounded below by some $\lambda_{lower} \in (0, n)$ when*

$$N \geq N_{min} := \frac{1}{\log q} \log \left(\frac{4n^2 - 4n - 2n\lambda_{lower} - \tau(n)}{6n^2 - 8n} \right),$$

where

$$\tau(n) := \sqrt{8n(2-n)\lambda_{lower}^2 + 8n^2(n-2)\lambda_{lower} + 4n^2(n-2)^2}.$$

Proof: Define $\hat{q} = (1-p)^N$. Then from Theorem 3.1 we find

$$n(1-\hat{q}) - \sqrt{2n(n-2)}\sqrt{\hat{q}(1-\hat{q})} \geq \lambda_{lower}.$$

Rearranging terms and then squaring both sides, we find

$$\Big(n(1-\hat{q}) - \lambda_{lower}\Big)^2 \geq 2n(n-2)(1-\hat{q})\hat{q}. \tag{3.9}$$

Expanding the squared term and gathering like terms, one finds

$$\big(3n^2 - 4n\big)\hat{q}^2 + \big(2n\lambda_{lower} - 4n^2 + 4n\big)\hat{q} + \big(n^2 - 2n\lambda_{lower} + \lambda_{lower}^2\big) \geq 0. \tag{3.10}$$

We seek here to make $1-(1-p)^N$ large, which is done by making $\hat{q} = (1-p)^N$ small. Using the

quadratic equation, we solve the inequality in Equation (3.10) to derive the lower bound

$$\hat{q} \leq \frac{4n^2 - 4n - 2n\lambda_{lower} - \tau(n)}{6n^2 - 8n},$$

where taking logarithms gives

$$N \log q \leq \log \left(\frac{4n^2 - 4n - 2n\lambda_{lower} - \tau(n)}{6n^2 - 8n} \right).$$

From $p \in (0, 1)$, we have $q \in (0, 1)$ and thus that $\log q < 0$. Then we lower-bound N via

$$N \geq \frac{1}{\log q} \log \left(\frac{4n^2 - 4n - 2n\lambda_{lower} - \tau(n)}{6n^2 - 8n} \right).$$

■

The last result needed before solving Problems 3.1 and 3.2 is the Paley-Zygmund inequality, which we state below.

Lemma 3.11 (Paley-Zygmund Inequality) *Let Z be a non-negative random variable satisfying* $\text{Var}[Z] < \infty$*, and let $\theta \in [0, 1]$. Then*

$$\mathbb{P}(Z > \theta \mathbb{E}[Z]) \geq (1 - \theta)^2 \frac{\mathbb{E}[Z]^2}{\mathbb{E}[Z^2]}.$$

Proof: See [71]. ■

We now state our solutions to Problems 3.1 and 3.2, beginning with Problem 3.1. Below, we use the results of Lemma 3.6 and Corollary 3.1, which show that although a complete graph has $\lambda_2 = n$, one cannot have $\mathbb{E}[\lambda_2] = n$. Therefore, we state our solution to Problem 3.1 in terms of approximating this value.

Theorem 3.3 *(Solution to Problem 3.1) Let $\varepsilon \in (0, n)$ be given. For $n \in \mathbb{N}$ nodes and edge probability $p \in (0, 1)$, a union of N random graphs from $\mathcal{G}(n, p)$ has $\mathbb{E}[\lambda_2(U_N)] \geq n - \varepsilon$ for*

values of N satisfying

$$N \geq N_{min} := \frac{1}{\log q} \log \left(\frac{4n^2 - 4n - 2n(n-\varepsilon) - \tau(n)}{6n^2 - 8n} \right),$$

where

$$\tau(n) := \sqrt{8n(2-n)(n-\varepsilon)^2 + 8n^2(n-2)(n-\varepsilon) + 4n^2(n-2)^2}.$$

In addition, for $N \geq N_{min}$, the probability that $\lambda_2(U_N)$ is at least $n-\varepsilon$ is bounded below according to

$$\mathbb{P}[\lambda_2(U_N) \geq n - \varepsilon] \geq \left(1 - \frac{(n-\varepsilon)}{n\hat{p}}\right)^2 \left(\frac{\left(n\hat{p} - \sqrt{2n(n-2)\hat{p}\hat{q}}\right)^2}{n(n-2)\hat{p}^2 + 2n\hat{p}} \right),$$

where $\hat{p} = 1 - (1-p)^N$ and $\hat{q} = 1 - \hat{p}$.

Proof: The first part of the theorem follows directly from Theorem 3.2, with $\lambda_{lower} = n - \varepsilon$. The second part of the theorem follows from the Paley-Zygmund inequality with

$$\theta = \frac{n - \varepsilon}{\mathbb{E}[\lambda_2(U_N)]},$$

which is in $[0, 1]$ because we consider unions with $N \geq N_{min}$ graphs in them. ■

Next, we present our solution to Problem 3.2.

Theorem 3.4 *(Solution to Problem 3.2) Let λ_{min} be the minimum algebraic connectivity of a connected graph on n nodes. Then for $n \in \mathbb{N}$ nodes and edge probability $p \in (0, 1)$, a union of N random graphs from $\mathcal{G}(n, p)$ has $\mathbb{E}[\lambda_2(U_N)] \geq \lambda_{min}$ for values of N satisfying*

$$N \geq N_{min} := \frac{1}{\log q} \log \left(\frac{4n^2 - 8n + 4n\cos\frac{\pi}{n} - \tau(n)}{6n^2 - 8n} \right),$$

where

$$\tau(n) := \sqrt{32(2-n)\left(1 - \cos\frac{\pi}{n}\right)^2 + 8n^2(n-2)\left(1 - \cos\frac{\pi}{n}\right)^2 + 4n^2(n-2)^2}.$$

In addition, for $N \geq N_{min}$, the probability that $\lambda_2(U_N)$ is at least λ_{min} is bounded below according to

$$\mathbb{P}\left[\lambda_2(U_N) \geq 2\left(1-\cos\frac{\pi}{n}\right)\right] \geq \left(1-\frac{2\left(1-\cos\frac{\pi}{n}\right)}{n\hat{p}}\right)^2 \left(\frac{\left(n\hat{p}-\sqrt{2n(n-2)\hat{p}\hat{q}}\right)^2}{n(n-2)\hat{p}^2+2n\hat{p}}\right),$$

where $\hat{p} = 1-(1-p)^N$ and $\hat{q} = 1-\hat{p}$.

Proof: This follows from the same procedure as Theorem 3.3. ■

A useful feature of Theorems 3.3 and 3.4 is that the expectation of λ_2 can be tuned to a particular value by modulating the number of graphs in the union. Thus, both λ_{lower} and N provide a means for manipulating the bounds in these theorems in order to derive probabilistic bounds on the algebraic connectivity of a union of random graphs. The next section implements these results for several values of N, n, and p.

3.4 Simulation Results

In this section, we simulate the results of Theorems 3.3 and 3.4 to provide numerical solutions to Problems 3.1 and 3.2 for select values N, n, and p.

3.4.1 Numerical Results for Problem 3.1

We begin by presenting results using Theorem 3.3. First, we give values of N_{min} for a range of values of n and p to give numerical values of N_{min} for different use cases. Table 3.1 gives the value of N_{min}, rounded up to the nearest integer, as determined by Theorem 3.3. The value of n ranges from 10 to $100,000$, and p ranges from 0.001 to 0.3. The values of N_{min} shown below are the numbers of random graphs needed in a union before the expected algebraic connectivity of that union is at least $n-\varepsilon$, where here we have selected $\varepsilon = 1$.

Throughout these values, we see that increasing n causes substantial increases in the number of graphs required for completeness. Completeness of a graph is a strong property in that it requires every edge to be present and, because a complete graph on n nodes contains $\frac{n(n-1)}{2}$ edges, increas-

Table 3.1: Values of N_{min}, as determined by Theorem 3.3.

p \ n	10	100	1,000	10,000	100,000
0.001	5,183	9,889	14,501	19,105	23,708
0.01	516	985	1,444	1,902	2,361
0.1	50	94	138	182	226
0.2	24	45	66	86	107
0.3	15	28	41	54	67

ing n requires more edges to be present. When each edge appears only randomly, having them *all* appear necessarily requires many trials, hence more graphs are required in a union and thus N_{min} increases with n.

Using the second half of Theorem 3.3, we next present lower bounds on the probability of completeness for unions of random graphs in Table 3.2. In particular, for unions of $N = 10$ graphs, we provide lower-bounds on $\mathbb{P}[\lambda_2(U_{10}) \geq n - \varepsilon]$ for n ranging from 3 to 24 and p ranging from 0.3 to 0.5. Here we have again selected $\varepsilon = 1$.

Table 3.2: A lower bound on the probability of $N = 10$ graphs from $\mathcal{G}(n, p)$ being complete, as determined by Theorem 3.3.

n \ p	0.30	0.40	0.50
3	0.3173	0.3857	0.4214
6	0.0630	0.0896	0.1028
9	0.0245	0.0385	0.0453
12	0.0123	0.0211	0.0253
15	0.0070	0.0132	0.0161
18	0.0043	0.0090	0.0112
21	0.0028	0.0064	0.0082
24	0.0019	0.0048	0.0062

Similar to the results in Table 3.1, Table 3.2 shows that the probability of completeness rapidly decreases as the number of nodes in a graph increases. Intuitively, this makes sense as a larger graph requires more total edges to be present in order to attain completeness, and thus we expect a larger graph to be complete less often. Here, it may be possible to attain tighter bounds on these probabilities, though our use of the Paley-Zygmund inequality gives a general-purpose result for

bounding these probabilities, and a range of lower-bounds on $\lambda_2(U_N)$ can be used with this same result. Along these lines, we next present numerical results from Theorem 3.4.

3.4.2 Numerical Results for Problem 3.2

We now present results using Theorem 3.4 by providing values of N_{min} for a range of values of n and p, representing the solutions to Problem 3.2 under different conditions. Table 3.3 gives the value of N_{min} (rounded up) as determined by Theorem 3.4 for n ranging from 10 to $100,000$ and p ranging from 0.00001 to 0.1. The values of N_{min} shown in the table are the numbers of graphs from $\mathcal{G}(n,p)$ needed in a union before the algebraic connectivity of the union has expectation bounded below by that of a line graph (which has least algebraic connectivity among all connected graphs).

Table 3.3: Values of N_{min}, as determined by Theorem 3.4.

p \ n	10	100	$1,000$	$10,000$	$100,000$
0.00001	$117,846$	$110,539$	$109,928$	$109,868$	$109,862$
0.0001	$11,785$	$11,054$	$10,993$	$10,987$	$10,986$
0.001	$1,178$	$1,105$	$1,099$	$1,099$	$1,099$
0.01	118	110	110	110	110
0.1	12	11	11	11	11

Throughout these values, it can be seen that, for each fixed value of n, an order of magnitude increase in p corresponds well to an order of magnitude decrease in the number of graphs required for connectivity of a union. In addition, for a fixed value of p, increasing n causes a decrease in the lower bound on N_{min}. This means that, as graphs become larger, fewer total graphs are required in a union to make it connected. This occurs because a graph on n nodes has $\frac{n(n-1)}{2}$ possible edges and, because a larger graph has more possible edges, larger graphs have more possible ways to attain connectivity, resulting in fewer required in a union to make it connected. The limiting behavior of Theorem 3.4 can be seen in Table 3.3, where the lower bounds on N_{min} appear to saturate when p is held fixed and n is increased.

In Theorem 3.4, the numerator of the second $\log$ term approaches $2n^2$, while the denominator of the same term approaches $6n^2$, causing this term to limit to $\log(1/3)$. Then, for large values of

n, the value of N_{min} in Theorem 3.4 is

$$N_{min} \approx \frac{\log(1/3)}{\log q}.$$

For $p = 0.00001$, this is equal to $109,861$, which agrees very closely with the values of N_{min} seen in Table 3.3 for $n = 10,000$ and $n = 100,000$.

For $n = 50$ nodes and $p \in \{0.05, 0.10, 0.15, 0.20, 0.25\}$, we now present lower bounds on the probability of a union of $N = 50$ random graphs being connected. Below, Table 3.4 gives numerical values for the probability of a particular union of $N = 50$ random graphs having algebraic connectivity bounded below by that of a line graph.

Table 3.4: A lower bound on the probability of $N = 50$ graphs from $\mathcal{G}(50, p)$ being connected, as determined by Theorem 3.4.

n \ p	0.05	0.10	0.15	0.20	0.25
50	0.359	0.810	0.953	0.989	0.998

In Table 3.4 we see that the probability of having a connected union increases rapidly as p increases, and that this probability approaches 1 even when p is far from 1. Next, in Table 3.5, we present results that have fixed values of $n = 50$ and $p = 0.10$, but changing values of N.

Table 3.5: A lower bound on the probability of N graphs from $\mathcal{G}(50, 0.1)$ being connected, as determined by Theorem 3.4.

n \ N	25	50	75	100	125
50	0.377	0.810	0.947	0.986	0.996

Similar to what was seen in Table 3.4, these results show that the probability of a union being connected increases rapidly with N. To further illustrate this trend, a union of $N = 250$ graphs with $n = 50$ and $p = 0.1$ is connected with probability at least 0.9998 according to Theorem 3.4.

The numerical results in this section indicate that the results in Theorems 3.3 and 3.4 solve Problems 3.1 and 3.2 in a manner which readily provides numerical results. The three parameters n, p, and N can vary dramatically across problem formulations, though the results obtained here

apply to broad ranges across all three of these parameters, allowing for these results to be used in a wide range of applications.

3.5 Conclusion

This chapter presented several results concerning the connectivity and completeness of unions of random graphs. In contrast to some established literature on random graphs, the graphs considered were on a fixed number of nodes, and the probability of any edge being present was not assumed to have any relationship to the number of nodes. The first four moments of a random graph's Laplacian were computed and, using these moments, bounds on the expectation and variance of the algebraic connectivity of unions of random graphs were derived. These bounds were then used to compute a lower bound on the number of random graphs needed in a union to attain some lower bound for the algebraic connectivity of the union. These results give a worst-case analysis for the rate at which cycles occur in Chapter 2. Given the utility and flexibility of random graphs in modeling multi-agent communications, this chapter gives a general-purpose result for the timing with which one can expect cycles to occur in Algorithm 2.2 and thus enables a general-purpose error estimate for Algorithm 2.2 in networks with asynchronous communications.

CHAPTER 4

ASYNCHRONOUS COORDINATION OF ROBOTIC NETWORKS

This chapter presents a robotic implementation of the asynchronous optimization algorithm developed in Chapter 2. This chapter serves the dual purposes of verifying the preceding results in a practical setting and demonstrating that this algorithm is indeed well-suited to implementation on a physical system. Below, Section 4.1 describes the problems being solved, Section 4.2 describes the robots being used and implementation details of the experimental setup, and Section 4.3 presents results from these robotic experiments.

4.1 Optimization Problems for Quadrotor Teams

The experiments in this chapter centered on solutions to three optimization problems which are crafted with specific physical formations in mind. They are stated here in the order in which they are executed in the experimental run described in Section 4.3, where they are solved by a team of five quadrotors. In all three problems, agent i has a state vector in $\mathbb{R}^3$. When writing the arguments of functions below, $x_{i,p}$ refers to the p^{th} entry of agent i's state vector.

First, Problem 4.1 has the agents form a regular pentagon in the xy-plane and evenly space their altitudes along the z-direction.

Problem 4.1

$$\begin{aligned} &\text{minimize } f(x) \\ &\text{subject to } g(x) \leq 0 \\ &x \in X, \end{aligned}$$

where

$$f(x) = \frac{1}{2}\Big(\|x_1 - x_2 - d_1\|^2 + \|x_2 - x_3 - d_2\|^2 + \|x_3 - x_4 - d_3\|^2 \\ + \|x_4 - x_5 - d_4\|^2 + \|x_5 - x_1 - d_5\|^2 \Big),$$

$$d_1 = \begin{pmatrix} -0.5 \\ 0.5 \\ 0.25 \end{pmatrix}, d_2 = \begin{pmatrix} 0.5 \\ 0.5 \\ 0.25 \end{pmatrix}, d_3 = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}, d_4 = \begin{pmatrix} 0.5 \\ -0.5 \\ 0.25 \end{pmatrix}, d_5 = \begin{pmatrix} -0.5 \\ -0.5 \\ 0.25 \end{pmatrix},$$

$$g(x) = \begin{pmatrix} \frac{1}{8}\left(\|x_1 - x_2\|^2 - \frac{1}{4}\right) \\ \frac{1}{8}\left(\|x_2 - x_3\|^2 - \frac{1}{4}\right) \\ \frac{1}{8}\left(\|x_3 - x_4\|^2 - \frac{1}{4}\right) \\ \frac{1}{8}\left(\|x_4 - x_5\|^2 - \frac{1}{4}\right) \\ \frac{1}{8}\left(\|x_5 - x_1\|^2 - \frac{1}{4}\right) \end{pmatrix},$$

and

$$X = \prod_{i \in [5]} [-0.8, 0.8] \times [-1.1, 1.1] \times [-2.5, -0.4].$$

♦

Second, Problem 4.2 has the agents form a "V"-shaped formation by forming a line in the xy-plane and evenly spacing their altitudes along the z-direction.

Problem 4.2

$$\text{minimize } f(x)$$

$$\text{subject to } g(x) \leq 0$$

$$x \in X,$$

where

$$f(x) \;=\; \frac{1}{2}\big(\|x_1 - d_2\|^2 + \|x_2 - d_2\|^2 + \|x_3 - d_3\|^2 + \|x_4 - d_4\|^2 + \|x_5 - d_5\|^2\big),$$

$$d_1 = \begin{pmatrix} 0.5 \\ 0 \\ -1 \end{pmatrix}, d_2 = \begin{pmatrix} 0 \\ -0.5 \\ -1.5 \end{pmatrix}, d_3 = \begin{pmatrix} 1 \\ 0.5 \\ -1.5 \end{pmatrix}, d_4 = \begin{pmatrix} -0.5 \\ -1 \\ -2 \end{pmatrix}, d_5 = \begin{pmatrix} 1.5 \\ 1 \\ -2 \end{pmatrix},$$

$$g(x) = \begin{pmatrix} (\|x_1 - x_2\|^2 - 1) \\ (\|x_1 - x_3\|^2 - 1) \\ (\|x_3 - x_5\|^2 - 1) \\ (\|x_2 - x_4\|^2 - 1) \end{pmatrix},$$

and

$$X = \prod_{i \in [5]} [-0.8, 0.8] \times [-1.1, 1.1] \times [-2.5, -0.4].$$

♦

Third, Problem 4.3 has the agents form a line in the xy-plane at the maximum permissible altitude as defined by the ensemble set constraint X.

Problem 4.3

$$\text{minimize } f(x)$$

$$x \in X,$$

where

$$f(x) \;=\; 5\big((x_{1,3} + 2)^2 + (x_{2,3} + 2)^2 + (x_{3,3} + 2)^2 + (x_{4,3} + 2)^2 + (x_{5,3} + 2)^2\big),$$

$$d_1 = \begin{pmatrix} 0.5 \\ 0 \\ -1 \end{pmatrix}, d_2 = \begin{pmatrix} 0 \\ -0.5 \\ -1.5 \end{pmatrix}, d_3 = \begin{pmatrix} 1 \\ 0.5 \\ -1.5 \end{pmatrix}, d_4 = \begin{pmatrix} -0.5 \\ -1 \\ -2 \end{pmatrix}, d_5 = \begin{pmatrix} 1.5 \\ 1 \\ -2 \end{pmatrix},$$

and

$$X = \prod_{i \in [5]} [-0.8, 0.8] \times [-1.1, 1.1] \times [-2.5, -0.4].$$

♦

4.2 Experimental Platform and Setup

The experiments in this chapter were conducted using five Crazyflie 2.0[1] quadrotors. Each quadrotor was running the Robot Operating System (ROS)[2] and communicated with a desktop computer functioning as a base station. This base station gathered information from 10 OptiTrack[3] motion capture cameras, giving real-time three-dimensional position information for each quadrotor. This camera system abstracts away the problems of sensing and state estimation for robot positions, letting Algorithm 2.2 instead take accurate information as a given, and focusing this implementation of Algorithm 2.2 on verification of the algorithm itself.

A quadrotor's dynamics are complex [72], though they are differentially flat [73], which means that quadrotor trajectories can be planned for a subset of the quadrotor's states, and all necessary inputs and other states can be computed in terms of these states and their derivatives. In particular, letting x, y, and z denote the position of the quadrotor in space, and letting ψ denote its yaw angle, by choosing

$$\boldsymbol{r} = \begin{pmatrix} x \\ y \\ z \\ \psi \end{pmatrix}$$

[1] http://www.bitcraze.io/crazyflie-2/
[2] http://www.ros.org/
[3] http://optitrack.com/

one can compute each rotor's thrust, the quadrotor's roll, and the quadrotor's pitch in terms of $\boldsymbol{r}$ $\dot{\boldsymbol{r}}$, $\ddot{\boldsymbol{r}}$, $\dddot{\boldsymbol{r}}$, and $\ddddot{\boldsymbol{r}}$ [73]. In this experimental setup, desired robot positions are generated in $\mathbb{R}^3$ and a trajectory between the current position and the desired one is generated using the framework in [73] to translate a desired trajectory into rotor thrusts. A problem is regarded as solved when the cloud determines that

$$\left\| \frac{\partial f}{\partial x}\big(x^c(t)\big) + \frac{\partial g}{\partial x}\big(x^c(t)\big)^T \mu(t) + \alpha x^c(t) \right\| < \varepsilon_{norm},$$

where $\varepsilon_{norm} = 0.20$ was chosen in this case. When this condition is satisfied, the agents begin solving the next problem in the sequence, or, if they have completed solving the final problem, they hold their final position until the experiment is terminated by a user.

To implement this approach, the aforementioned base station gathers global information about the network and therefore serves the role of the cloud computer in this setting. The hardware limitations of this platform do not allow for direct robot-to-robot communications, and these communications are instead simulated by storing each robot's knowledge of the network's state onboard the base station and ensuring that each agent's knowledge is segregated from each other agent's; though controlling the quadrotors is technically a centralized operation in this experimental setup, the information restrictions and discrepancies seen in practice under asynchronous communications are present in this implementation because we separate the agents' knowledge of the network. The base station generates subsequent positions for each agent using Algorithm 2.2 and the agents move to these locations using local PID controllers on their positions in space. Due to disturbances in the environment, each agent will not necessarily reach exactly its commanded position, and the agents' true positions are therefore fed back into Algorithm 2.2 for determining subsequent positions.

There are three additional modifications made to Algorithm 2.2 for implementation. The first is the use of a collision avoidance system for the quadrotors. Algorithm 2.2 intentionally allows states of two agents to be arbitrarily close (or indeed, to overlap exactly) because this is desirable in some

settings and because a safe radius around each agent is heavily implementation dependent. For this implementation, we add a basic collision avoidance system that uses control barrier functions (CBFs) [74] to provide minimally-invasive modifications to each quadrotor's position commands in order to prevent collisions between them. In this case, a safety radius of 30cm was chosen in order to keep the centers of mass of any two quadrotors at least 30cm apart.

Second, to prevent excessively many commands from being sent to the quadrotors and to prevent the jitter that can result from sending too many commands in too short a span of time, the cloud only commands the quadrotors to move if such a command will move the quadrotors at least 30cm total. If a command would not result in at least 30cm of total displacement, the agents are stationary while new positions are repeatedly computed until the 30cm threshold is reached.

Third, to make the behavior of the system truly asynchronous, all agents' communications were randomized. During each iteration of Algorithm 2.2, each agent computed a state update with probability $p_{update} = 0.1$, and each agent sent its current state value (whether it was just updated or not) to each essential neighbor and the cloud with probability $p_{send} = 0.10$. Here, all communications and computations were independent, giving rise to asynchronous behavior in the system.

For Algorithm 2.2, the regularization parameters $\alpha = 0.02$ and $\beta = 0.02$ were chosen, and the primal and dual stepsizes were chosen to be

$$\gamma = \frac{2}{L_p + \alpha}$$

and

$$\rho = 0.99\rho_0,$$

where ρ_0 is defined in Theorem 2.1.

4.3 Experimental Results

We now present results from an experimental implementation of the above setup. The total experiment duration was 83.3s. The time required to solve each problem is given in Table 4.1.

Table 4.1: The experimental time required for five quadrotors to asynchronously solve Problems 4.1, 4.2, and 4.3 using Algorithm 2.2.

Task	Runtime (s)
Solve Problem 4.1	51.1
Solve Problem 4.2	23.8
Solve Problem 4.3	8.4

Problem 4.1 took longest to solve in part because it has the most complex functional constraints g; Problem 4.2 has simpler constraints and thus takes less time, and Problem 4.3 does not have functional constraints and it is the fastest to solve.

The initial position of the quadrotors is shown below in Figures 4.1 and 4.2 from the side and from above, respectively.

Figure 4.1: A side view of the starting positions of five quadrotors running Algorithm 2.2.

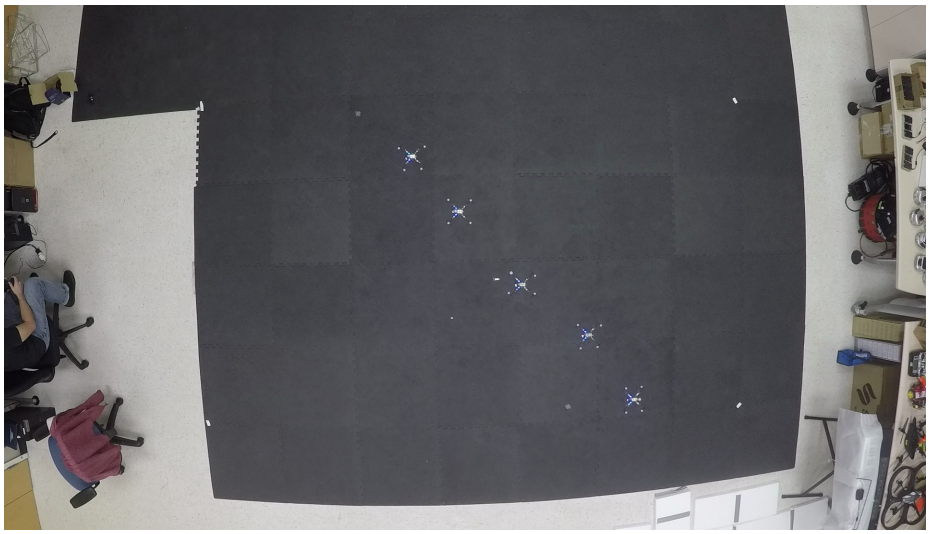

Figure 4.2: A top view of the starting positions of five quadrotors running Algorithm 2.2.

After 51.1 seconds, the quadrotors arrive at their solution to Problem 4.1, shown from both the side and from above in Figures 4.3 and 4.4, respectively. Here we see that the quadrotors indeed arrive at a pentagon in the xy-plane and arrive at evenly-space positions in the z-dimension, as expected.

Figure 4.3: A side view of the quadrotors' collective solution to Problem 4.1. As expected, they are evenly spaced in the z-dimension.

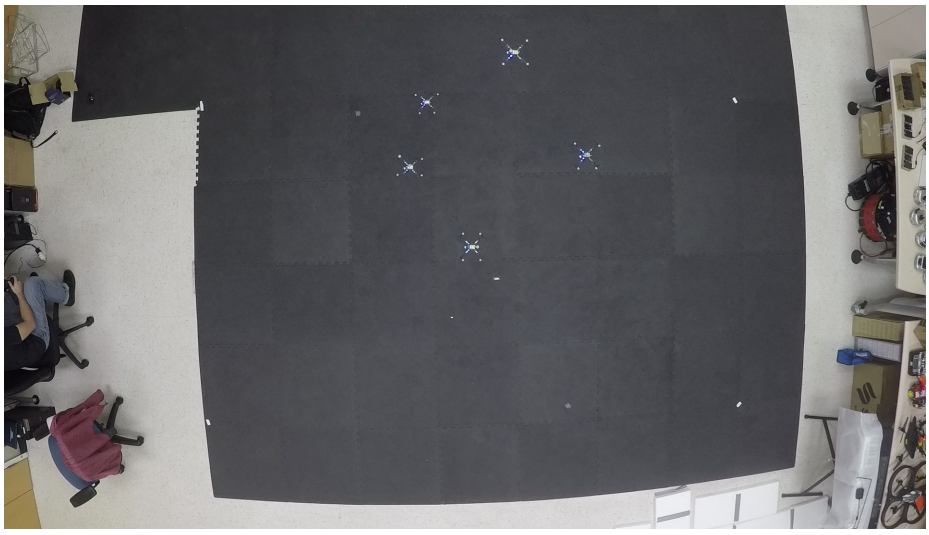

Figure 4.4: A top view of the quadrotors' collective solution to Problem 4.1. As expected, they form a pentagon in the xy-plane.

Once this solution is attained, the quadrotors begin solving Problem 4.2. After 23.8 seconds they reach the position shown in Figures 4.5 and 4.6. As described above, the quadrotors reach a "V" shape by assembling a line in the xy-plane and by attaining even spacing in the z-direction.

Figure 4.5: A side view of the quadrotors' solution to Problem 4.2. As expected, they are evenly spaced in the z-dimension and form a line in the xy-plane, giving rise to an overall "V" shape.

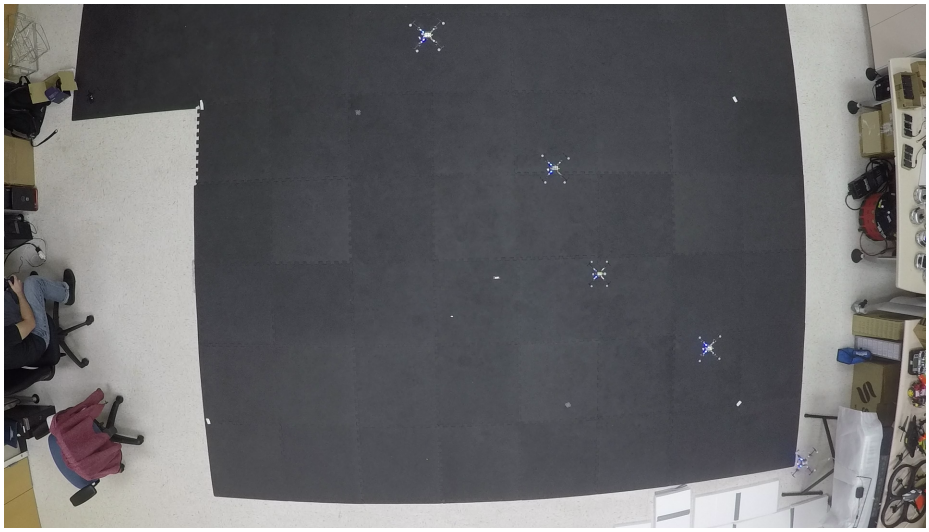

Figure 4.6: A top view of the quadrotors' solution to Problem 4.2. As expected, they are evenly spaced in the z-dimension and form a line in the xy-plane, giving rise to an overall "V" shape.

Finally, once the quadrotors solve Problem 4.2, they solve Problem 4.3. The solution to this problem is attained in 8.4 seconds and is shown from the side in Figure 4.7, and from above in Figure 4.8. As expected, because the cost only affects agents' z-coordinates, the agents form a straight line in the xy-plane with equal heights in the z-direction.

Figure 4.7: A side view of the quadrotors' collective solution to Problem 4.3. As encoded in Problem 4.3, the agents reach their maximum allowed heights while forming a line in the xy-plane.

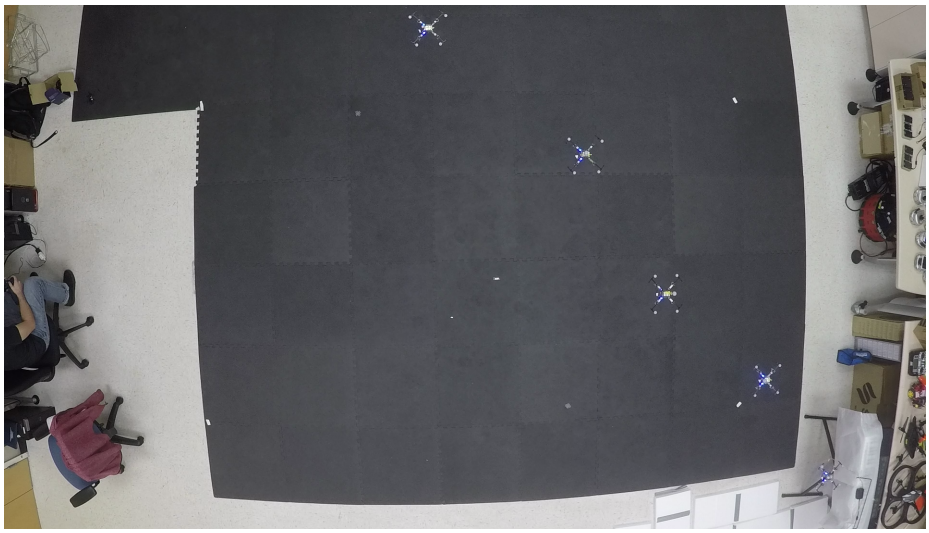

Figure 4.8: A top view of the quadrotors' collective solution to Problem 4.3. As encoded in Problem 4.3, the agents reach their maximum allowed heights while forming a line in the xy-plane.

4.4 Discussion and Conclusions

In this chapter, a robotic implementation of the work in Chapter 2 was presented, and it was shown that a team of five quadrotors can successfully use Algorithm 2.2 to solve practical network coordination tasks. The preceding experimental results show not only that Algorithm 2.2 is readily implementable in practice, but also that it can successfully solve problems even when communications are highly unreliable. This point is furthered by the fact that the robots' communications and computations were deliberately programmed to occur infrequently and at random times, yet the results of these experiments are visibly correct.

This work also demonstrates that Algorithm 2.2 is flexible enough to allow for certain necessary practical considerations, like collision avoidance as was done here using barrier functions. In spite of adding collision avoidance into the system, and in spite of the resulting position errors stemming from this collision avoidance implementation, Algorithm 2.2 still successfully solved Problems 4.1, 4.2, and 4.3, indicating that Algorithm 2.2 is indeed well-suited to practical applications.

Part II

Private Coordination

CHAPTER 5

DIFFERENTIALLY PRIVATE MULTI-AGENT OPTIMIZATION

Optimization problems spread across teams of agents arise naturally in several fields, including communications [9], [10], robotics [3], [6], machine learning [26], sensor networks [1], [2], and smart power grids [7], [8], [65]. Correspondingly, a variety of approaches have been developed that solve problems with a wide variety of formulations. For example, [75] allows for distributed optimization of non-differentiable objectives with time-varying communication links, [76] considers a similar problem formulation in which communication links fail over time, and [77] uses a distributed Newton method to solve dynamic network utility maximization problems. Many other problem types and solution schemes exist in the literature, and a broad exposition of results can be found in [18].

In some cases multi-agent optimization is done using sensitive user data. A concrete example of such a case comes from smart power grids. In smart power grids, homeowners share their power usage information with others on the grid to allow network management, e.g., frequency regulation [65], and to minimize their own power costs. In some cases, the granular power usage data shared in smart grids can be used to infer sensitive details of users' personal lives [16], [17]. In particular, smart grid data can "provide a detailed breakdown of energy usage over a long period of time, which can show patterns of use," [17, Page 15, Item 16]. Further, given these patterns, "[p]rofiles can thus be developed and then applied back to individual households and individual members of these households," [17, Page 15, Item 18]. These usage patterns in turn "could reveal personal details about the lives of consumers, such as their daily schedules," [16, Page 2, Paragraph 5].

It is precisely the deduction of such patterns that we wish to prevent in the context of multi-agent optimization. Based on the potentially revealing nature of some user data, we seek to optimize while protecting sensitive user data both from eavesdroppers and other agents in the network.

In some sense, privacy and optimization are competing objectives in that agents who only seek to optimize may freely share their states with others in the network, while agents concerned only with privacy may be inclined to share no information at all. To privately optimize, then, we must strike a balance between these two different, competing objectives.

One approach to privacy that has recently seen widespread use is *differential privacy*. Differential privacy was originally established in the database literature and keeps sensitive database entries private when a database is queried by adding noise to the result of that query [78], [79]. The authors of [80], [81] survey some of the important developments in this vein. Differential privacy has been adapted to dynamical systems in order to keep sensitive inputs private from an adversary observing a system's outputs [82]. A dynamical system is differentially private if inputs that are close in the input space produce outputs that have similar probability distributions; these notions will be made precise in Section 5.2.

It is the dynamical systems notion of differential privacy that we apply to keep agents' state trajectories private while optimizing. One appealing aspect of differential privacy is its resilience to post-processing, which allows for arbitrary processing of private information without the threat of its privacy guarantees being weakened [82, Theorem 1]. Differential privacy is also robust to arbitrary side information, meaning that an adversary cannot weaken differential privacy by much through using information gleaned from another source [83].

There has already been some work on enforcing differential privacy in optimization. In [84] linear programs are solved in a framework that allows for keeping objective functions or constraints private. The authors of [85] consider a similar setting wherein linearly constrained problems with affine objectives are solved while keeping the objective functions private. In the multi-agent setting, [86] solves distributed consensus-type problems while keeping the agents' objective functions private, while [87] solves similar problems while keeping each agent's initial state private.

In this chapter we solve non-linear programs wherein each agent's state trajectory is sensitive information and the agents therefore seek to protect their exact state trajectories from other agents and any eavesdroppers. To protect these sensitive data, a trusted cloud computer is used that

performs certain computations upon information it receives from the agents, makes the results of those computations private by adding noise to them, and then sends the private results to each agent. Each agent then updates its state locally using the information it received from the cloud, and this process of sharing and updating information is repeated. This algorithm is online in the sense that agents' state trajectories are not first planned and then executed; instead, each agent determines subsequent state values in terms of its present state. The contribution of this chapter thus consists of an online multi-agent optimization algorithm that solves constrained optimization problems while keeping each agent's state trajectory differentially private. With this algorithm, we also provide probabilistic convergence rates and relate the level of privacy in the system to the convergence of the algorithm.

Our motivation for developing a mixed centralized/decentralized algorithm is inspired by the prominence of cloud computing in many real-world applications. A survey of existing cloud applications is given in [88], and that reference elaborates on the scalability of the cloud and its ability to coordinate many mobile devices. It is precisely these features of the cloud that make it an attractive choice here. In this chapter, the cloud, viewed as a central aggregator, is an integral part of the optimization process, and we leverage its scalability to aggregate all agents' information, perform computations upon that data in a private manner, and then distribute these private results to the agents.

The privacy implementation in this chapter differs from the aforementioned references on private optimization in several key ways. We are interested in solving problems in which the agents collectively run an on-line optimization algorithm collaboratively by sharing (private functions of) sensitive information. In the problems we consider, each iteration of the optimization algorithm determines each agent's next state. That is, the iterates of the optimization algorithm *are* the agents' states, and it is each agent's desire to keep its state trajectory private to protect information about its behavior. Therefore, while the aforementioned references on private optimization focus on privacy for other types of problem data, we focus specifically on keeping entire state trajectories private while optimizing, and do so using the framework for trajectory-level privacy put forth in [82]. In

addition, we incorporate both nonlinear inequality constraints and set constraints, which, to our knowledge, has not been explored in other privacy implementations.

Given the need to optimize while remaining private, encryption alone cannot provide the privacy guarantees that are needed in the problems we examine. In the "upstream" direction, encryption could be used to protect communications sent from the agents to the cloud, provided the cloud could decrypt them. However in the "downstream" direction, when the cloud sends transmissions to the agents, any encrypted messages from the cloud would naturally need to be decrypted by the agents to allow each agent to update its state. While this strategy can protect transmissions of sensitive data from eavesdroppers, having the agents decrypt transmissions from the cloud would expose all agents' sensitive data to each other agent in the network, violating the privacy guarantees that are required. Instead, what is required here is a privacy implementation that protects user data from eavesdroppers *and* all others in the network, while still making that data useful for optimizing. It is for this reason that we use differential privacy.

The rest of the chapter is organized as follows. Section 5.1 defines the problem to be solved and its method of solution. Next, Section 5.2 covers the necessary elements of differential privacy and relates them to the setting of optimization. Then, Section 5.3 provides a proof of convergence for the optimization algorithm used here and a bound on its convergence, in addition to exploring the trade-off between privacy and convergence. Section 5.4 provides simulation results to support the theoretical developments made. Finally, Section 5.5 provides concluding remarks.

5.1 Optimization Problem Formulation

In this section we define the problem to be solved. In Section 5.1.1 we define the multi-agent problem of interest and then, to aid in the exposition of its solution method, we formulate an equivalent ensemble problem. Then the solution to that problem will be discussed and, in Section 5.1.2, will be adapted to the cloud-based architecture used here. Throughout this section, the term "ensemble problem" refers to a centralized problem that is equivalent to the problem solved by the agents and cloud, which is not centralized. The ensemble problem and multi-agent problem are equivalent (in

that they lead to the same solution), though discussion will be carried out in terms of the ensemble problem for simplicity.

5.1.1 Problem Overview

Consider N agents indexed over the set $I := \{1, \ldots, N\}$, with agent i having state $x_i \in \mathbb{R}^{n_i}$ for some $n_i \in \mathbb{N}$. Agent i seeks to minimize the objective function

$$f_i : \mathbb{R}^{n_i} \to \mathbb{R},$$

where f_i depends only upon x_i, that is, each agent's objective function has no dependence upon the other agents' states. Using the notation $\nabla_i f_i := \frac{\partial f_i}{\partial x_i}$, we state the following assumption for objective functions.

Assumption 5.1 *The function f_i is C^1 and convex, and $\nabla_i f_i$ is Lipschitz with constant L_i for all $i \in I$.* $\lozenge$

Assumption 5.1 allows for a broad class of functions to be used as objective functions, including any C^2 convex function on a compact, convex domain (cf. Assumption 5.2 below). Each agent's state is constrained to lie in a given set which we express as

$$x_i \in X_i \subseteq \mathbb{R}^{n_i}.$$

Regarding each set X_i, we state the following assumption.

Assumption 5.2 *Each set X_i is non-empty, compact, and convex.* $\lozenge$

In particular Assumption 5.2 admits box constraints which are common in some multi-agent

problems. Now define the ensemble state vector

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix},$$

where $n = \sum_{i=1}^{N} n_i$. We impose global inequality constraints on the agents by requiring

$$g(x) := \begin{pmatrix} g_1(x) \\ \vdots \\ g_m(x) \end{pmatrix} \leq 0,$$

where the above inequality is enforced component-wise, i.e., $g_j(x) \leq 0$ for all $j \in J := \{1, \ldots, m\}$. We now state our assumptions on g.

Assumption 5.3 *The function $g_j : \mathbb{R}^n \to \mathbb{R}$ is C^1 and convex for all $j \in J$. In addition, for $\mathbb{R}^m$ and $\mathbb{R}^n$ both equipped with the same p-norm, the function $g_{x_j} := \frac{\partial g}{\partial x_j}$ is Lipschitz continuous with constant K_p^j for all $j \in J$ with respect to the metric induced by the p-norm. The function g is Lipschitz with constant K_p^g with respect to the same metric.* $\lozenge$

In this chapter we focus on the cases of $p = 1$ and $p = 2$. Like Assumption 5.1, Assumption 5.3 allows for any convex, C^2 functions to be used for constraints whenever Assumption 5.2 holds. We also have the following assumption on g.

Assumption 5.4 *The constraints satisfy Slater's condition, namely there exists a point $\bar{x} \in X$ such that $g(\bar{x}) < 0$.* $\lozenge$

Assumption 5.4 is commonly enforced in nonlinear programming problems to guarantee that strong duality holds. Under Assumptions 5.1-5.4, we state an ensemble-level optimization problem. To do so, we define the ensemble objective

$$f(x) = \sum_{i=1}^{n} f_i(x_i)$$

and the ensemble constraint set

$$X = \prod_{i=1}^{n} X_i,$$

where the product is meant in the Cartesian sense. To fix ideas, we state the following optimization problem that does not yet incorporate privacy; privacy will be formally included in Problem 5.3 in Section 5.2.

Problem 5.1 (Preliminary; no privacy requirement)

$$\begin{aligned} &\text{minimize } f(x) \\ &\text{subject to } g(x) \leq 0 \\ &x \in X. \end{aligned}$$

♦

We note here that Problem 5.1 will be solved without having agent i share f_i or X_i with the other agents or with the cloud because these data are considered sensitive information. Similarly, g is considered sensitive and the cloud does not share g or any g_{x_i} with any of the agents.

The Lagrangian associated with Problem 5.1 is

$$L(x, \mu) = f(x) + \mu^T g(x),$$

where μ is a vector of Kuhn-Tucker multipliers in the non-negative orthant of $\mathbb{R}^m$, denoted $\mathbb{R}^m_+$. Under Assumptions 5.1, 5.2, and 5.3 a primal solution $\hat{x}$ exists and the set of all primal solutions is non-empty and compact. With the addition of Assumption 5.4, a dual solution $\hat{\mu}$ exists and the optimal primal and dual values are equal [28, Proposition 6.4.3].

Under Assumptions 5.1-5.4, a point $\hat{x}$ solves Problem 5.1 if and only if there exists a point $\hat{\mu} \in \mathbb{R}^m_+$ such that $(\hat{x}, \hat{\mu})$ is a saddle point of L, that is, if and only if the point $(\hat{x}, \hat{\mu})$ satisfies

$$L(\hat{x}, \mu) \leq L(\hat{x}, \hat{\mu}) \leq L(x, \hat{\mu})$$

for all $(x, \mu) \in X \times \mathbb{R}^m_+$ [28, Proposition 6.2.4]. It is as saddle points of L that we seek solutions $(\hat{x}, \hat{\mu})$ to Problem 5.1.

Toward doing so, we next define the symbols

$$L_x := \frac{\partial L}{\partial x},\ L_\mu := \frac{\partial L}{\partial \mu}, g_x := \frac{\partial g}{\partial x},\ \text{and } f_x := \frac{\partial f}{\partial x},$$

and define the map

$$G(x, \mu) = \begin{pmatrix} L_x(x, \mu) \\ -L_\mu(x, \mu) \end{pmatrix}.$$

In what follows, it is necessary for G to be a Lipschitz mapping. Though the maps f_x and g_x are Lipschitz by Assumptions 5.1 and 5.3, G itself cannot be shown to be Lipschitz because its domain, $X \times \mathbb{R}^m_+$, is unbounded by virtue of $\mathbb{R}^m_+$ being unbounded. To rectify this situation, we find a non-empty, convex, compact set containing $\hat{\mu}$ as was done in [31]. By the saddle point condition on $(\hat{x}, \hat{\mu})$, we have $L(\hat{x}, \hat{\mu}) \leq L(x, \hat{\mu})$ for all $x \in X$. Expanding, we find

$$f(\hat{x}) + \hat{\mu}^T g(\hat{x}) \leq f(\bar{x}) + \hat{\mu}^T g(\bar{x})$$

for $\bar{x}$ the Slater point as defined in Assumption 5.4. By the complementary slackness condition we have

$$f(\hat{x}) \leq f(\bar{x}) + \hat{\mu}^T g(\bar{x}).$$

Rearranging we find

$$\sum_{j=1}^{m} \hat{\mu}_j \leq \frac{f(\bar{x}) - f(\hat{x})}{\min\limits_{1 \leq j \leq m} \{-g_j(\bar{x})\}} \leq \frac{f(\bar{x}) - f(x^*)}{\min\limits_{1 \leq j \leq m} \{-g_j(\bar{x})\}},$$

where $x^* \in \arg\min_{x \in X} f(x)$. We then define the set

$$\mathbb{M} := \left\{ \mu \in \mathbb{R}^m_+ : \|\mu\|_1 \leq \frac{f(\bar{x}) - f(x^*)}{\min\limits_{1 \leq j \leq m} \{-g_j(\bar{x})\}} \right\},$$

which is non-empty, compact, and convex by definition, and which contains $\hat{\mu}$. For economy of notation, we define the symbols $Z := X \times \mathbb{M}$ and $\hat{z} := (\hat{x}, \hat{\mu})$, and we will use $z := (x, \mu)$ to denote an arbitrary point in Z. Below, we use the notions of monotonicity and strong monotonicity for operators, which we define now.

Definition 5.1 *An operator $F : \mathbb{R}^n \to \mathbb{R}^n$ is said to be* monotone *if $\big(F(x) - F(y)\big)^T (x - y) \geq 0$ for all $x, y \in \mathbb{R}^n$, and* strongly monotone *if, for some $\eta > 0$, $\big(F(x) - F(y)\big)^T (x - y) \geq \eta \|x - y\|^2$ for all $x, y \in \mathbb{R}^n$.* $\Diamond$

By definition, every strongly monotone map is also monotone. Since $L(\cdot, \mu)$ is convex for all $\mu \in \mathbb{M}$ and $L(x, \cdot)$ is concave for all $x \in X$, we see that G is monotone [89, Theorem A]. Under Assumptions 5.1-5.4, a primal-dual pair $(\hat{x}, \hat{\mu})$ is a saddle point of L if and only if it solves the following variational inequality (VI) [90, Corollary 11.1].

Problem 5.2 *(VI formulation; no privacy requirement yet) Find a point $\hat{z} \in Z$ such that*

$$\langle z - \hat{z}, G(\hat{z}) \rangle \geq 0$$

for all $z \in Z$. $\blacklozenge$

Further discussion on the equivalence of Problems 5.1 and 5.2 is given in [30], Sections 1.3.1, 1.3.2, and 1.4.1. Privacy is formally added to Problem 5.2 in the statement of Problem 5.3 in Section 5.2.

We will use the notation $VI(K, F)$ to denote the generic problem of finding a point $x \in K$ such that, for a monotone map F, $\langle y - x, F(x) \rangle \geq 0$ for all $y \in K$, and we will use the notation $SOL(K, F)$ to denote the solution set of $VI(K, F)$. The symbols Z and G refer to the specific problem under consideration in this chapter so that Problem 5.2 is denoted $VI(Z, G)$ and its solution set is $SOL(Z, G)$. It is in the setting of variational inequalities that we will proceed and we focus on solving Problem 5.2 with the understanding that its solutions also solve Problem 5.1.

For a compact set K and a monotone map F, one method of solving the variational inequality $VI(K, F)$ is using a projection method with an iterative Tikhonov regularization as was done

for deterministic variational inequalities in [34] and for stochastic variational inequalities in [91]; these methods regularize the earlier Goldstein-Levitin-Polyak method for solving such problems [92], [93]. The basic principle underlying these methods is that a point in $SOL(K, F)$ can be approached iteratively with F specifying the direction in which to move at each iteration. To endow this procedure with greater numerical stability and, as will be shown, robustness to noise, the k^{th} iteration specified in [34], [91] instead uses the direction specified by $F + \alpha_k I$, with I the identity map, $\alpha_k > 0$, and $\alpha_k \to 0$. When F is monotone, each map $F + \alpha_k I$ is strongly monotone so that $SOL(K, F + \alpha_k I)$ is a singleton for all k. Letting ξ_k denote the (unique) element of $SOL(K, F + \alpha_k I)$, for $\alpha_k > 0$ and $\alpha_k \to 0$ we have $\xi_k \to y_0$, where y_0 is the least-norm element of $SOL(K, F)$ (which itself is non-empty because K is compact and F is monotone).

Given an initial point $z(0) \in Z$, the deterministic form of the regularized method to solve $VI(Z, G)$ is given below in Algorithm 5.1.

Algorithm 5.1 *Given a point $z(0) \in Z$, apply the update*

$$\begin{aligned} z(k+1) &= \Pi_Z \left[z(k) - \gamma_k \left(G\big(z(k)\big) + \alpha_k z(k) \right) \right] \\ &= \Pi_Z \left[\begin{pmatrix} x(k) - \gamma_k \left(f_x\big(x(k)\big) + g_x\big(x(k)\big)^T \mu(k) + \alpha_k x(k) \right) \\ \mu(k) + \gamma_k \left(g\big(x(k)\big) - \alpha_k \mu(k) \right) \end{pmatrix} \right] \end{aligned} \tag{5.1}$$

until a fixed point $\hat{z}$ is reached. $\triangle$

Here α_k is the regularization parameter at timestep k and γ_k is the step-size at the same timestep. In Section 5.2 we will use Algorithm 5.1 to solve a private optimization problem, and in Section 5.3 we provide hypotheses on γ_k and α_k sufficient for convergence. Currently we show the applicability of this style of solution to the cloud architecture mentioned above.

5.1.2 Communications

If we separate the update law in Algorithm 5.1 to examine the per-agent primal update law, we find that agent i executes

$$x_i(k+1) = \Pi_{X_i}\left[x_i(k) - \gamma_k\left(\nabla_i f_i(x_i(k)) + g_{x_i}(x(k))^T\mu(k) + \alpha_k x_i(k)\right)\right]. \tag{5.2}$$

The only terms on the right-hand side of this update law that contain information from other agents are $g_{x_i}\big(x(k)\big)$ and $\mu(k)$. Though g_{x_i} is a function of all states in the network, the agents do not send their states to each other directly to allow for its computation because doing so may reveal sensitive information. Instead, every agent sends its state to a trusted cloud computer which computes $g_{x_i}(x(k))$ for every $i \in I$. Because no agent has every agent's state value, no agent can compute $\mu(k)$ (cf. Equation (5.1)), and therefore the cloud computes $\mu(k)$ as well using the update law

$$\mu(k+1) = \Pi_{\mathbb{M}}\left[\mu(k) + \gamma_k\left(g\big(x(k)\big) - \alpha_k\mu(k)\right)\right].$$

Then, to use Algorithm 5.1 with this architecture, the cloud sends a private form of $g_{x_i}\big(x(k)\big)^T\mu(k)$ to agent i; the modifications to these quantities to make them private are covered in Section 5.2. The cloud is assumed to be a powerful computer capable of carrying out these calculations quickly so that they reliably arrive at the agents in a timely fashion.

With this communications scheme, at timestep k four actions occur. First, agent i sends $x_i(k)$ to the cloud and the cloud assembles all agents' states into the vector $x(k)$. Second, the cloud computes $g_{x_i}\big(x(k)\big)$ for all $i \in I$ in a differentially private way. Third, the cloud sends a private form of $g_{x_i}\big(x(k)\big)^T\mu(k)$ to agent i. Fourth, agent i computes $x_i(k+1)$ while the cloud simultaneously computes $\mu(k+1)$ in a differentially private way, and then this sequence of communications and computations is repeated. Because this happens at every timestep, information in the network is always synchronized when computations occur and there is no disagreement among the agents or cloud as to what the value of a particular state is. As a result, the computations that are spread

across the network in this manner produce identical results to Algorithm 5.1, and the ensemble problem is mathematically equivalent to the cloud-based multi-agent problem.

For simplicity, the forthcoming analysis will be carried out in the ensemble setting. Despite the mathematical equivalence between the multi-agent and ensemble approaches, the advantage of the cloud-based approach in practice is that it allows for each agent's state trajectory to be kept private while the ensemble approach does not.

5.2 Private Optimization

Differential privacy originates in the database literature in computer science and was originally designed to keep individual entries of a database private [81]. It has recently been extended to the setting of dynamical systems in [82]. Differential privacy offers a formal definition of privacy as well as resilience to post-processing and robustness to side information. This resilience to post-processing prevents an adversary from weakening the guarantees of differential privacy by performing post-hoc calculations on private information. Robustness to side information guarantees that an adversary cannot use information it has gleaned from an alternate source to fully defeat differential privacy. Below we first review differential privacy, then give a formal private optimization problem statement, and finally discuss applying privacy to Problem 5.2.

5.2.1 Differentially Private Systems

Let there be N input signals to a system, each contributed by some user. The i^{th} input signal is denoted u_i and is contained in the set $\tilde{\ell}_{p_i}^{s_i}$, namely the space of sequences of s_i-vectors equipped with the p_i norm, with $s_i, p_i \in \mathbb{N}$, such that every finite truncation of u_i is in $\ell_{p_i}^{s_i}$. More explicitly, let $u_i(k)$ denote the k^{th} element of u_i and define

$$P_T u_i = \begin{cases} u_i(k) & \text{for } k \leq T \\ 0 & \text{otherwise.} \end{cases}$$

Then we say $u_i \in \tilde{\ell}^{s_i}_{p_i}$ if and only if $P_T u_i$ has finite p_i-norm for all values of T. Using this definition, the full input space to the system is

$$\tilde{\ell}^s_p = \prod_{i=1}^{N} \tilde{\ell}^{s_i}_{p_i},$$

where the product is meant in the Cartesian sense, and the system produces outputs $y \in \tilde{\ell}^r_q$. In this chapter we consider the cases where $p_i = 1$ for all $i \in I$ or $p_i = 2$ for all $i \in I$. In the case of $p_i = 1$, the full input space to the system is $\tilde{\ell}^s_1$ and we use the ordinary 1-norm on this space. For $p_i = 2$, we likewise use the ordinary 2-norm on $\tilde{\ell}^s_2$. While each of $\|\cdot\|_1$ and $\|\cdot\|_2$ will be used for both the 1-norm and 2-norm on $\mathbb{R}^n$ and $\tilde{\ell}^s_p$, the intent of each symbol can be discerned from its argument each time it is used.

To implement differential privacy, we must specify which inputs we wish to generate "similar" outputs. To do this, fix a real number $B > 0$ and define the binary symmetric adjacency relation $\text{Adj}_B : \tilde{\ell}^s_p \times \tilde{\ell}^s_p \to \{0, 1\}$ as

$$\text{Adj}_B(u, \tilde{u}) = 1 \Leftrightarrow \|u - \tilde{u}\|_p \leq B. \tag{5.3}$$

Two inputs u and $\tilde{u}$ for which $\text{Adj}(u, \tilde{u}) = 1$ are called "adjacent."

Towards making precise the notion of "similar" outputs, fix a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, and let $\mathcal{B}^d$ denote the Borel σ-algebra on $\mathbb{R}^d$. Differential privacy is enforced by a *mechanism*, which is a map M taking the form

$$M : \tilde{\ell}^s_p \times \Omega \to \tilde{\ell}^r_q,$$

and the role of a mechanism is to approximate a system whose inputs are sensitive information. We now state the definition of a differentially private mechanism. In this definition, we use a σ-algebra over $\tilde{\ell}^r_q$, denoted $\Sigma_{q,r}$.[1]

Definition 5.2 *A mechanism $M : \tilde{\ell}^s_p \times \Omega \to \tilde{\ell}^r_q$ is (ε, δ)-differentially private if and only if, for all*

[1] An explicit construction of this σ-algebra can be found in [82, Section III-A], though we avoid a lengthy exposition on $\Sigma_{q,r}$ due to the relatively minor role its technical details play in the current work.

adjacent $u, \tilde{u} \in \tilde{\ell}_p^s$ *we have*

$$\mathbb{P}(M(u) \in S) \leq e^{\varepsilon}\mathbb{P}(M(\tilde{u}) \in S) + \delta \tag{5.4}$$

for all $S \in \Sigma_{q,r}$. $\lozenge$

In Equation (5.4) it is ε and δ that determine the privacy policy and smaller values of each imply a greater level of privacy for users. In general, ε should be kept small and typical values for ε range from 0.1 to $\log 3$. On the other hand, δ should be kept as small as possible because it allows for zero probability events for $M(\tilde{u})$ to have non-zero probability for $M(u)$, and therefore can allow for important losses in privacy by making it easy for an adversary to distinguish between outputs. Common values for δ range from 0 to 0.05; $(\varepsilon, 0)$-differential privacy is called ε-differential privacy and, in general, ε-differential privacy is stronger than (ε, δ)-differential privacy precisely because of the aforementioned losses in privacy that can come from $\delta > 0$. For this reason, (ε, δ)-differential privacy can be regarded as a δ-approximate form of ε-differential privacy [94]. For a fixed value of ε, the benefit of using even small values of $\delta > 0$ is that the variance of noise added can be reduced while maintaining "almost" the same level of privacy.

5.2.2 Private Optimization Problem Statement

In the setting of Problem 5.2, we want to protect the state trajectory $\boldsymbol{x} = \big(x(k)\big)_{k\in\mathbb{N}}$, which is a sensitive signal in $\tilde{\ell}_p^s$, and in doing so we protect each individual agent's state trajectory; for agent i, this is $\boldsymbol{x}_i \in \tilde{\ell}_{p_i}^{s_i}$. As discussed in Section 5.1, keeping individual agents' state trajectories private is necessary when the cloud computes g_{x_i} and μ at each time k. To implement privacy in these computations, we regard each g_{x_i} as a deterministic, causal, memory-less dynamical system and seek to make each such system differentially private. Similarly, we regard g as a deterministic, causal, memory-less dynamical system and seek to make it differentially private as well. Due to the post-processing property of differential privacy, computing μ using a private form of g also implies that μ keeps each $\boldsymbol{x}_i$ private.

As discussed in Section 5.1, the agents do not communicate with each other at all and, instead, each agent only sends its state to the cloud. The cloud handles all required centralized computations and sends (privatized forms of) their results to the agents. Let $\hat{g}_{x_i}$ denote the private form of g_{x_i}, let $\hat{g}$ denote the private form of g, and let $\hat{\mu}$ denote a dual vector μ that has been computed using $\hat{g}$ instead of g.

At time k the cloud sends to agent i the vector

$$\hat{p}_i(k) = \hat{g}_{x_i}\big(x(k)\big)^T \hat{\mu}(k).$$

We are interested in having a team of agents optimize by having the cloud send agent i only $\hat{p}_i(k)$ at time k. We require that $\hat{p}_i(k)$ protect $\boldsymbol{x}_i$ for all $i \in I$, and we implement privacy by approximating g_{x_i} (for all $i \in I$) and g by differentially private mechanisms. Using this method of communications, we state the following problem that incorporates both optimization and privacy objectives, and respects the fact that the objectives and constraints in this problem are sensitive data.

Problem 5.3 *(Private optimization) Solve Problem 5.2 using Algorithm 5.1 while*

i. the agents communicate only with the cloud (i.e., there is no inter-agent communication)

ii. the cloud makes the systems g and g_{x_i}, $i \in I$ (whose inputs are the agents' state trajectories) differentially private in the sense of Definition 5.2

iii. agent i does not share f_i or X_i with any other agent or the cloud

iv. the cloud does not share g or any g_{x_i} with any agent. ♦

Points iii and iv in Problem 5.3 are required because f_i, X_i, g_{x_i}, and g are considered sensitive information. These data could also be useful to an adversary attempting to infer agents' states using the system dynamics, and it is beneficial not to share these problem data for precisely this reason. Towards solving Problem 5.3, we now review mechanisms which implement differential privacy for dynamical systems.

5.2.3 Privacy-Preserving Mechanisms

To define a mechanism for enforcing differential privacy, we must first also define the sensitivity of a system, which is used to determine the variance of noise that must be added in a privacy-preserving mechanism. Letting $\mathcal{G}$ be a deterministic causal system, the sensitivity of $\mathcal{G}$ is an upper bound on the distance between $\mathcal{G}(u)$ and $\mathcal{G}(\tilde{u})$ whenever $\text{Adj}_B(u, \tilde{u}) = 1$ holds. Formally we define the ℓ_p sensitivity of $\mathcal{G}$, denoted $\Delta_p\mathcal{G}$, as

$$\Delta_p\mathcal{G} := \sup_{u,\tilde{u}:\text{Adj}_B(u,\tilde{u})=1} \|\mathcal{G}(u) - \mathcal{G}(\tilde{u})\|_p.$$

The mechanism we will use for ε-differential privacy is the Laplace mechanism, which adds noise drawn from a Laplace distribution. Below we use the notation $\text{Lap}(\mu, b)$ to denote the Laplace distribution with mean μ and scale parameter b.

Theorem 5.1 *([82, Theorem 4]) Let the adjacency relation defined in Equation* (5.3) *be used with* $p = 1$ *and let* $\mathcal{G}$ *be a system with sensitivity* $\Delta_1\mathcal{G}$*. Let a privacy parameter* $\varepsilon \geq 0$ *be given and recall that* r *is the dimension of the output space. Then the mechanism* $M(x) = \mathcal{G}(x) + w$ *where* $w(k) \sim \text{Lap}(0, b/\varepsilon)^r$ *and* $b \geq \Delta_1\mathcal{G}$ *is* ε*-differentially private.* ■

For (ε, δ)-differential privacy, we will use the Gaussian mechanism. Its definition requires that we first define $\kappa(\delta, \varepsilon)$ using the $\mathcal{Q}$-function,

$$\mathcal{Q}(y) := \frac{1}{\sqrt{2\pi}} \int_y^\infty e^{-\frac{v^2}{2}} dv.$$

The function $\kappa(\delta, \varepsilon)$ is defined for $\varepsilon \geq 0$ and $0 < \delta < \frac{1}{2}$ as

$$\kappa(\delta, \varepsilon) := \frac{1}{2\varepsilon}\big(K_\delta + \sqrt{K_\delta^2 + 2\varepsilon}\big),$$

where $K_\delta = \mathcal{Q}^{-1}(\delta)$. We now define the Gaussian mechanism.

Theorem 5.2 *([82, Theorem 3]) Let the adjacency relation defined in Equation* (5.3) *be used with* $p = 2$ *and let* $\mathcal{G}$ *be a system with sensitivity* $\Delta_2\mathcal{G}$*, with privacy parameters* $\varepsilon \geq 0$ *and* $0 < \delta < \frac{1}{2}$ *given and* r *the dimension of the output space. Then the mechanism* $M(x) = \mathcal{G}(x) + w$ *where* $w(k) \sim \mathcal{N}(0, \sigma^2 I_r)$ *is* (ε, δ)*-differentially private for* $\sigma \geq \kappa(\delta, \varepsilon)\Delta_2\mathcal{G}$. ■

Theorems 5.1 and 5.2 provide a lower bound on the variance of each noise that is added, and we assume that these variances are also chosen to be finite. We now compute the sensitivities that are needed to implement differential privacy in Problem 5.3.

5.2.4 Computing Sensitivities

In Problem 5.3 it is desired to protect the value of $\boldsymbol{x}$, including from agents in the network. In the per-agent update law in Equation (5.2), $x(k)$ appears in g_{x_i}, and g_{x_i} must therefore be made private when the cloud computes $g_{x_i}(x(k))$. To protect $\boldsymbol{x}$ in this way, the cloud adds noise directly to $g_{x_i}\big(x(k)\big)$, and the variance of noise that must be added depends on the sensitivity of g_{x_i}. To compute the sensitivity of g_{x_i} we regard it as a memoryless dynamical system and generalize it to act on entire signals of states. Recalling that $x(k) \in X$ for all k, Assumption 5.2 provides that X is bounded, and therefore $x(k)$ is as well for all $k \in \mathbb{N}$. Then $\boldsymbol{x} \in \tilde{\ell}_p^n$.

We now overload the notation g_{x_i} by allowing it to act on elements of $\tilde{\ell}_p^n$. In particular, g_{x_i} acts on elements of $\mathbb{R}^n$ as before and for state trajectories $\boldsymbol{x} \in \tilde{\ell}_p^n$ we define

$$g_{x_i}(\boldsymbol{x}) := \big(g_{x_i}(x(k))\big)_{k\in\mathbb{N}}.$$

We now fix a real scalar $B > 0$. For two state trajectories, $\boldsymbol{x}, \tilde{\boldsymbol{x}} \in \tilde{\ell}_p^n$ such that $\mathrm{Adj}_B(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = 1$ holds, we compute the sensitivity of g_{x_i} according to

$$\begin{aligned}\Delta_p g_{x_i} &= \sup_{\boldsymbol{x},\tilde{\boldsymbol{x}}:\mathrm{Adj}_B(\boldsymbol{x},\tilde{\boldsymbol{x}})=1} \|g_{x_i}(\boldsymbol{x}) - g_{x_i}(\tilde{\boldsymbol{x}})\|_p \\ &\leq \sup_{\boldsymbol{x},\tilde{\boldsymbol{x}}:\mathrm{Adj}_B(\boldsymbol{x},\tilde{\boldsymbol{x}})=1} K_p^i \sqrt[p]{\sum_{k=0}^{\infty} \|x(k) - \tilde{x}(k)\|_p^p} \leq K_p^i B,\end{aligned}$$

where we have used $\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|_p \leq B$ and where this bound on the sensitivity holds for g_{x_i} for all $i \in I$.

In computing $\mu(k)$, the cloud must also add noise in some fashion because $\mu(k)$ depends upon $x(k)$. We regard g as a dynamical system and make it private, and the resilience of differential privacy to post-processing guarantees that $\boldsymbol{\mu} = \big(\mu(k)\big)_{k\in\mathbb{N}}$ keeps $\boldsymbol{x}$ private. To compute the sensitivity of g, we extend it to act on $\boldsymbol{x} \in \tilde{\ell}_p^n$ as above. For $\boldsymbol{x}, \tilde{\boldsymbol{x}} \in \tilde{\ell}_p^n$ satisfying $\mathrm{Adj}_B(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = 1$, we use the same procedure as was used above for g_{x_i} to find

$$\Delta_p g = \sup_{\boldsymbol{x},\tilde{\boldsymbol{x}}:\mathrm{Adj}_B(\boldsymbol{x},\tilde{\boldsymbol{x}})=1} \|g(\boldsymbol{x}) - g(\tilde{\boldsymbol{x}})\|_p \leq K_p^g B.$$

Having computed the requisite sensitivities, we return to solving Problem 5.3.

5.2.5 Optimizing in the Presence of Noise

We now examine how noise appears in Algorithm 5.1 once it has been added for privacy. For $g_{x_i}\big(x(k)\big)$ we add noise $w_i(k) \in \mathbb{R}^{m\times n_i}$ drawn from either a Laplace or Gaussian distribution and for $g\big(x(k)\big)$ we add noise $w_g(k) \in \mathbb{R}^m$ drawn from the same class of distribution as the w_i, with all noises independent. Define w_x by

$$w_x = (w_1 \; w_2 \; \cdots \; w_n) \in \mathbb{R}^{m\times n}.$$

In ensemble form the private dynamics under consideration are

$$\begin{aligned} z(k+1) &= \begin{pmatrix} x(k+1) \\ \mu(k+1) \end{pmatrix} \\ &= \Pi_Z \left[z(k) - \gamma_k \begin{pmatrix} f_x\big(x(k)\big) + \big(\frac{\partial g}{\partial x}\big(x(k)\big) + w_x(k)\big)^T \mu(k) + \alpha_k x(k) \\ -g\big(x(k)\big) + w_g(k) + \alpha_k \mu(k) \end{pmatrix} \right]. \end{aligned}$$

Expanding, we find

$$z(k+1) = \begin{pmatrix} x(k+1) \\ \mu(k+1) \end{pmatrix}$$
$$= \Pi_Z \left[z(k) - \gamma_k \begin{pmatrix} f_x\big(x(k)\big) + \frac{\partial g}{\partial x}\big(x(k)\big)^T \mu(k) + w_x(k)^T \mu(k) + \alpha_k x(k) \\ -g\big(x(k)\big) + w_g(k) + \alpha_k \mu(k) \end{pmatrix} \right].$$

Because $\mu(k) \in \mathbb{R}^m_+$, each element of $w_x(k)^T \mu(k)$ is some weighted combination of elements of $w_x(k)$ with non-negative weights. Combined with the independence of the noises used for privacy, this results in each entry of $w_x(k)^T \mu(k)$ being a random variable having variance that is the weighted sum of variances of elements of $w_x(k)$. With this in mind we define the random vector $w_s(k) = w_x(k)^T \mu(k)$ (which we note has finite variance since $\mu(k)$ is contained in $\mathbb{M}$ and $w_x(k)$ has finite variance), and zero mean (because $w_i(k)$ has zero mean for all $k \in \mathbb{N}$ and all $i \in I$). Then we find

$$z(k+1) = \Pi_Z \left[z(k) - \gamma_k \begin{pmatrix} f_x\big(x(k)\big) + \frac{\partial g}{\partial x}\big(x(k)\big)^T \mu(k) + w_s(k) + \alpha_k x(k) \\ -g\big(x(k)\big) + w_g(k) + \alpha_k \mu(k) \end{pmatrix} \right]$$
$$= \Pi_Z \left[z(k) - \gamma_k \left(G\big(z(k)\big) + \alpha_k z(k) + w(k) \right) \right],$$

where $w(k)$ denotes the noise added at timestep k and aggregates all noisy signals used for privacy. We state this stochastic update law as Algorithm 5.2.

Algorithm 5.2 *Given $z(0) \in Z$, apply the update law*

$$z(k+1) = \Pi_Z \left[z(k) - \gamma_k \left(G\big(z(k)\big) + \alpha_k z(k) + w(k) \right) \right]$$

until a fixed point $\hat{z} \in Z$ is reached. △

We note that by its definition $\mathbb{E}[w(k)] = 0$, and observe that this noisy update law is equivalent to Algorithm 5.1 with an additional noise term added. As discussed above, Algorithm 5.2 allows

the agents to optimize without requiring that agent i share f_i or X_i with any other agents or the cloud. In addition, as will be shown in Section 5.3, Algorithm 5.2 tolerates noise of constant variance, as required by differential privacy, while still allowing for convergence in mean-square. Convergence of Algorithm 5.2 is the subject of the next section.

5.3 Convergence of Private Optimization

In this section we prove the convergence of Algorithm 5.2. Algorithm 5.2 was first presented in [34] without noise and was presented in its noisy form in [91]. Both papers omit complete proofs and, due to the heavy dependence of this work upon Algorithm 5.2, we provide a proof here. To the best of our knowledge a proof of the convergence of Algorithm 5.2 as stated in [91] is not available in the literature; similar work is presented in [95], [96] which cover algorithms related to Algorithm 5.2, though those works impose additional assumptions upon α_k and γ_k due to the differences in the problems studied in those works.

5.3.1 Main Convergence Result

Now we explore in depth solving variational inequalities using a Tikhonov regularized projection method, the basic elements of which are covered in [30, Section 12.2]. Earlier it was stated that if $SOL(K, F) \neq \emptyset$, then for $\xi_k \in SOL(K, F + \alpha_k I)$ we have $\xi_k \to y_0$, where y_0 is the least-norm element of $SOL(K, F)$. Using that $\{\xi_k\}_{k\in\mathbb{N}}$ is a convergent sequence, we find that $\{\|\xi_k\|\}_{k\in\mathbb{N}}$ is bounded and, in particular, there is some M_ξ such that $\|\xi_k\| \leq M_\xi$ for all k, e.g., $\|\xi_k\| \leq \sup_{z\in Z} \|z\|$.

Using this fact, the following lemma relates points $z(k)$ generated by Algorithm 5.2 to successive solutions to the problems $VI(Z, G + \alpha_k I)$ (each with α_k held constant). Recalling that ξ_k is the unique solution to $VI(Z, G + \alpha_k I)$, we have the following result.

Lemma 5.1 *For all* $k \in \mathbb{N}$

$$\|z(k) - \xi_k\|^2 \leq (1 + \gamma_k\alpha_k)\|z(k) - \xi_{k-1}\|^2 + M_\xi^2 \left(\frac{\alpha_{k-1} - \alpha_k}{\alpha_k}\right)^2 \left(\frac{1 + \gamma_k\alpha_k}{\gamma_k\alpha_k}\right).$$

Proof: First note that because ξ_k solves $VI(K, G + \alpha_k I)$, we have

$$(\xi_{k-1} - \xi_k)^T \big(G(\xi_k) + \alpha_k \xi_k\big) \geq 0. \tag{5.5}$$

Similarly for ξ_{k-1} we find

$$(\xi_k - \xi_{k-1})^T \big(G(\xi_{k-1}) + \alpha_{k-1} \xi_{k-1}\big) \geq 0. \tag{5.6}$$

Summing Equations (5.5) and (5.6), and using the monotonicity of G gives

$$(\xi_{k-1} - \xi_k)^T (\alpha_k \xi_k - \alpha_{k-1} \xi_{k-1}) \geq 0.$$

Adding and subtracting $\alpha_k \xi_{k-1}$ inside the second set of parentheses then gives

$$\begin{aligned}(\xi_{k-1} - \xi_k)^T (\alpha_k I - \alpha_{k-1} I) \xi_{k-1} &\geq \alpha_k (\xi_{k-1} - \xi_k)^T (\xi_{k-1} - \xi_k) \\ &= \alpha_k \|\xi_{k-1} - \xi_k\|^2.\end{aligned}$$

Using the Cauchy-Schwarz inequality results in

$$\|\xi_{k-1} - \xi_k\| \leq \frac{|\alpha_{k-1} - \alpha_k|}{\alpha_k} M_\xi. \tag{5.7}$$

Expanding the term $\|z(k) + \xi_{k-1} - \xi_{k-1} - \xi_k\|^2$ and applying Equation (5.7) then gives

$$\begin{aligned}\|z(k) - \xi_k\|^2 \leq \|z(k) - \xi_{k-1}\|^2 + \left(\frac{\alpha_{k-1} - \alpha_k}{\alpha_k}\right)^2 M_\xi^2 \\ + 2\left(\frac{|\alpha_{k-1} - \alpha_k|}{\alpha_k}\right) M_\xi \|z(k) - \xi_{k-1}\|.\end{aligned} \tag{5.8}$$

For the third term on the right-hand side above we have

$$2\left(\frac{|\alpha_{k-1}-\alpha_k|}{\alpha_k}\right)M_\xi\|z(k)-\xi_{k-1}\| = 2\sqrt{\gamma_k\alpha_k}\|z(k)-\xi_{k-1}\|\left(\frac{|\alpha_{k-1}-\alpha_k|}{\alpha_k\sqrt{\gamma_k\alpha_k}}\right)M_\xi$$
$$\leq \gamma_k\alpha_k\|z(k)-\xi_{k-1}\|^2 + M_\xi^2\left(\frac{(\alpha_{k-1}-\alpha_k)^2}{\gamma_k\alpha_k^3}\right), \quad (5.9)$$

where we have used that fact that $a^2+b^2 \geq 2ab$ for $a, b \in \mathbb{R}$.

Substituting Equation (5.9) into Equation (5.8) gives the desired result. ■

The other lemma we need concerns the convergence of sequences of random variables and enables a Lyapunov-like argument to be made for their convergence.

Lemma 5.2 *([36], Lemma* 10*, Page* 49*) Let $v_0, \ldots, v_k$ be a sequence of independent random variables with $v_k \geq 0$ and $\mathbb{E}[v_0] < \infty$. Suppose that $\mathbb{E}[v_{k+1}] \leq (1-\tau_k)v_k + \sigma_k$, with*

$$0 \leq \tau_k \leq 1, \quad \sigma_k \geq 0, \quad \sum_{k=0}^{\infty}\tau_k = \infty, \quad \frac{\sigma_k}{\tau_k} \to 0.$$

Then $\mathbb{E}[v_k] \to 0$. If, in addition, we have $\sum_{k=0}^{\infty}\sigma_k < \infty$, then $v_k \to 0$ almost surely and

$$\mathbb{P}(v_j \leq \varepsilon \text{ for all } j \geq k) \geq 1 - \frac{1}{\varepsilon}\left(\mathbb{E}[v_k] + \sum_{i=k}^{\infty}\sigma_i\right).$$

■

We now prove the convergence of Algorithm 5.2.

Theorem 5.3 *Let Assumptions 5.1-5.4 hold. Suppose that $\gamma_k > 0$ and $\alpha_k > 0$ satisfy the following four conditions:*

$$\sum_{k=0}^{\infty}\gamma_k\alpha_k = \infty, \ \frac{\gamma_k}{\alpha_k} \to 0, \ \alpha_k \to 0, \ \text{ and } \frac{(\alpha_{k-1}-\alpha_k)}{\gamma_k\alpha_k^2} \to 0.$$

Then for noise signal w with $\mathbb{E}[w(k)] = 0$ and bounded variance for all $k \in \mathbb{N}$, for the update rule

$$z(k+1) = \Pi_Z\left[z(k) - \gamma_k\left(G\big(z(k)\big) + \alpha_k z(k) + w(k)\right)\right],$$

we have $\mathbb{E}[\|z(k) - z_0\|^2] \to 0$*, where* z_0 *is the least-norm solution to Problem 5.2.*

Let L_G *be the Lipschitz constant of* G*. If, in addition to the above, the sequence of terms*

$$\sigma_k := \left(1 - \gamma_k\alpha_k\left(2 - \gamma_k\alpha_k - \frac{\gamma_k}{\alpha_k}L_G^2 - 2\gamma_k L_G\right)\right) M_\xi^2 \left(\frac{\alpha_{k-1} - \alpha_k}{\alpha_k}\right)^2 \left(\frac{1 + \gamma_k\alpha_k}{\gamma_k\alpha_k}\right) + \gamma_k^2\mathbb{E}[\|w(k)\|^2]$$

is summable, then the convergence estimate

$$\mathbb{P}(\|z(j) - \xi_{j-1}\|^2 \leq \varepsilon \text{ for all } j \geq k) \geq 1 - \frac{1}{\varepsilon}\left(\mathbb{E}[\|z(k) - \xi_{k-1}\|^2] + \sum_{i=k}^{\infty}\sigma_i\right) \tag{5.10}$$

holds for all $\varepsilon > 0$*.*

Proof: It was established in Section 5.1.1 that $SOL(Z, G) \neq \emptyset$ so that $\xi_k \to z_0$, where z_0 is the least-norm element of $SOL(Z, G)$ and where ξ_k solves $VI(Z, G + \alpha_k I)$. We now show that $z(k+1) \to \xi_k$.

Because ξ_k solves $VI(Z, G + \alpha_k I)$ we have

$$\xi_k = \Pi_Z\left[\xi_k - \gamma_k\big(G(\xi_k) + \alpha_k\xi_k\big)\right].$$

Using the non-expansive property of the projection operator and taking the expectation of both

sides we find

$$\begin{aligned}
\mathbb{E}\big[\|z(k+1)-\xi_k\|^2\big] &= \mathbb{E}\Big[\Big\|\Pi_Z\Big[z(k)-\gamma_k\big(G(z(k))+\alpha_k z(k)+w(k)\big)\Big] \\
&\qquad\qquad -\Pi_Z\Big[\xi_k-\gamma_k\big(G(\xi_k)+\alpha_k\xi_k\big)\Big]\Big\|^2\Big] \\
&\le \mathbb{E}\Big[\|z(k)-\xi_k+\gamma_k(G(\xi_k)-G(z(k))) \\
&\qquad\qquad -\gamma_k\alpha_k(z(k)-\xi_k)-\gamma_k w(k)\|^2\Big] \\
&= \mathbb{E}\Big[\|z(k)-\xi_k\|^2-2\gamma_k(\xi_k-z(k))^T(G(\xi_k)-G(z(k))) \\
&\qquad\qquad -2\gamma_k\alpha_k\|z(k)-\xi_k\|^2+\gamma_k^2\|G(\xi_k)-G(z(k))\|^2 \\
&\le \|z(k)-\xi_k\|^2-2\gamma_k\alpha_k\|z(k)-\xi_k\|^2+\gamma_k^2\|G(\xi_k)-G(z(k))\|^2 \\
&\quad +2\gamma_k^2\alpha_k(G(\xi_k)-G(z(k)))^T(\xi_k-z(k)) \\
&\quad +\gamma_k^2\alpha_k^2\|\xi_k-z(k)\|^2+\gamma_k^2\mathbb{E}[\|w(k)\|^2]
\end{aligned}$$

where the last inequality follows from the monotonicity of G, and where the fact that $\mathbb{E}[w(k)]=0$ has caused all terms containing $w(k)$ except $\mathbb{E}[\|w(k)\|^2]$ to vanish.

Using the Cauchy-Schwarz inequality then gives

$$\begin{aligned}
\mathbb{E}[\|z(k+1)-\xi_k\|^2] &\le \|z(k)-\xi_k\|^2-2\gamma_k\alpha_k\|z(k)-\xi_k\|^2 \\
&\quad +\gamma_k^2\|G(\xi_k)-G(z(k))\|^2+2\gamma_k^2\alpha_k\|G(\xi_k)-G(z(k))\|\|\xi_k-z(k)\| \\
&\quad +\gamma_k^2\alpha_k^2\|\xi_k-z(k)\|^2+\gamma_k^2\mathbb{E}[\|w(k)\|^2].
\end{aligned}$$

Assumptions 5.1-5.3 and the compactness of $\mathbb{M}$ together imply that G is Lipschitz and, denoting its Lipschitz constant by L_G, we have

$$\mathbb{E}[\|z(k+1)-\xi_k\|^2] \le \gamma_k^2\mathbb{E}[\|w(k)\|^2]+\Big(1-2\gamma_k\alpha_k+\gamma_k^2L_G^2+2\gamma_k^2\alpha_kL_G+\gamma_k^2\alpha_k^2\Big)\|z(k)-\xi_k\|^2.$$

Defining

$$\theta_k := 1 - \gamma_k\alpha_k\left(2 - \gamma_k\alpha_k - \frac{\gamma_k}{\alpha_k}L_G^2 - 2\gamma_k L_G\right)$$

and

$$\rho_k := M_\xi^2\left(\frac{\alpha_{k-1} - \alpha_k}{\alpha_k}\right)^2\left(\frac{1 + \gamma_k\alpha_k}{\gamma_k\alpha_k}\right),$$

applying Lemma 5.1 then gives

$$\mathbb{E}[\|z(k+1) - \xi_k\|^2] \leq \theta_k(1 + \gamma_k\alpha_k)\|z(k) - \xi_{k-1}\|^2 + \theta_k\rho_k + \gamma_k^2\mathbb{E}[\|w(k)\|^2]. \tag{5.11}$$

By hypothesis we have

$$\gamma_k\alpha_k \to 0, \quad \gamma_k/\alpha_k \to 0, \quad \gamma_k \to 0,$$

with $\alpha_k > 0$ and $\gamma_k > 0$ for all k. Then there exists an $M > 0$ such that for all $k \geq M$ we have both

$$\gamma_k\alpha_k \in (0, 1)$$

and

$$0 \leq 1 - 2\gamma_k\alpha_k \leq \theta_k \leq 1 - \gamma_k\alpha_k.$$

Then for all $k \geq M$, $\theta_k(1 + \gamma_k\alpha_k) \leq \theta_k + \gamma_k\alpha_k$ and thus for all $k \geq M$

$$\theta_k(1 + \gamma_k\alpha_k) \leq 1 - \gamma_k\alpha_k(1 - \gamma_k\alpha_k - \frac{\gamma_k}{\alpha_k}L_G^2 - 2\gamma_k L_G) \in (0, 1).$$

In particular, take some $\theta \in (0, 1)$ such that

$$1 - \gamma_k\alpha_k\left(1 - \gamma_k\alpha_k - \frac{\gamma_k}{\alpha_k}L_G^2 - 2\gamma_k L_G\right) \leq 1 - \gamma_k\alpha_k\theta$$

for all $k \geq M$. Then $1 - \gamma_k\alpha_k\theta \in (0, 1)$. Setting $\tau_k = \gamma_k\alpha_k\theta$ and $\sigma_k = \rho_k\theta_k + \gamma_k^2\mathbb{E}[\|w(k)\|^2]$ we

rewrite Equation (5.11) as

$$\mathbb{E}[\|z(k+1)-\xi_k\|^2] \leq (1-\tau_k)\|z(k)-\xi_{k-1}\|^2 + \sigma_k. \tag{5.12}$$

All that remains is to show that the conditions of Lemma 5.2 are met. First, $\tau_k \in (0,1)$ by construction. For all $k \geq M$ we have $\rho_k \geq 0$ and $\theta_k \geq 0$ so that $\sigma_k \geq 0$. Regarding summability of τ_k we find $\sum_{k=M}^{\infty} \tau_k = \theta \sum_{k=M}^{\infty} \gamma_k \alpha_k = \infty$ by hypothesis. To show that $\sigma_k/\tau_k \to 0$ we have

$$\begin{aligned}\frac{\sigma_k}{\tau_k} &= \frac{\theta_k \left(\frac{\alpha_{k-1}-\alpha_k}{\alpha_k}\right)^2 \left(\frac{1+\gamma_k\alpha_k}{\gamma_k\alpha_k}\right) M_\xi^2 + \gamma_k^2 \mathbb{E}[\|w(k)\|^2]}{\gamma_k \alpha_k \theta} \\ &= (1 - 2\gamma_k\alpha_k + \gamma_k^2\alpha_k^2 + \gamma_k^2 L_G^2 + 2\alpha_k\gamma_k^2 L_G) M_\xi^2 \left(\frac{\alpha_{k-1}-\alpha_k}{\gamma_k\alpha_k^2}\right)^2 (1+\gamma_k\alpha_k)\frac{1}{\theta} + \frac{1}{\theta}\frac{\gamma_k}{\alpha_k}\mathbb{E}[\|w(k)\|^2].\end{aligned}$$

Using the hypotheses regarding γ_k and α_k we have

$$1 - 2\gamma_k\alpha_k + \gamma_k^2\alpha_k^2 + \gamma_k^2 L_G^2 + 2\alpha_k\gamma_k^2 L_G \to 1$$

and

$$1 + \gamma_k\alpha_k \to 1,$$

along with

$$\left(\frac{\alpha_{k-1}-\alpha_k}{\gamma_k\alpha_k^2}\right)^2 \to 0,$$

so that the first term in $\frac{\sigma_k}{\tau_k}$ goes to zero. It was established in Section 5.2.5 that $\mathbb{E}[\|w(k)\|^2]$ is bounded above for all k, namely that $\mathbb{E}[\|w(k)\|^2] \leq K_w$ for some $K_w > 0$. Because $\frac{\gamma_k}{\alpha_k} \to 0$ we have $\frac{\gamma_k}{\alpha_k} K_w \to 0$ and hence $\frac{\sigma_k}{\tau_k} \to 0$ as desired, and the first part of the theorem follows from Lemma 5.2.

When the sequence $\{\sigma_k\}_{k\in\mathbb{N}}$ is summable, the additional convergence rate estimate is a straightforward application of Lemma 5.2 as well. ■

One valid choice of γ_k and α_k satisfying the conditions in Theorem 5.3 is

$$\alpha = \bar{\alpha} k^{-c_1} \text{ and } \gamma = \bar{\gamma} k^{-c_2},$$

with $\bar{\alpha} > 0$, $\bar{\gamma} > 0$, $0 < c_1 < c_2$, and $c_1 + c_2 < 1$ [91].

5.3.2 Convergence Rate Estimates

For the above choices of γ_k and α_k, we derive bounds on c_1 and c_2 which are sufficient to make σ_k summable. As shown in the proof of Theorem 5.3, there exists an $M > 0$ such that for all $k \geq M$ we have $\theta_k \in (0, 1)$, so that for all $k \geq M$ we have

$$\sigma_k \leq M_\xi^2 \left(\frac{\alpha_{k-1} - \alpha_k}{\alpha_k} \right)^2 \left(\frac{1 + \gamma_k \alpha_k}{\gamma_k \alpha_k} \right) + \gamma_k^2 \mathbb{E} \left[\|w(k)\|^2 \right]. \tag{5.13}$$

To make the second term in Equation (5.13) summable, we can set $\gamma_k = \bar{\gamma} k^{-c_2}$ with $c_2 > \frac{1}{2}$. Again using K_w to denote an upper bound on the variance of $w(k)$ gives

$$\sum_{k=1}^{\infty} \gamma_k^2 \mathbb{E}[\|w(k)\|^2] \leq \bar{\gamma}^2 K_w \zeta(2c_2), \tag{5.14}$$

where $\zeta(\cdot)$ is the Riemann zeta function [97], defined as

$$\zeta(p) = \sum_{n=1}^{\infty} \frac{1}{n^p},$$

which takes finite values for arguments $p > 1$.

Regarding the first term in Equation (5.13), we note that there is some $\hat{M} > 0$ such that

$$1 \leq \frac{1}{\bar{\alpha}\bar{\gamma} k^{-c_1} k^{-c_2}}$$

for all $k \geq \hat{M}$, and therefore

$$\left(\frac{\alpha_{k-1}-\alpha_k}{\alpha_k}\right)^2\left(\frac{1+\gamma_k\alpha_k}{\gamma_k\alpha_k}\right) = \left(\frac{\alpha_{k-1}-\alpha_k}{\alpha_k}\right)^2\left(1+\frac{1}{\gamma_k\alpha_k}\right) \leq 2\left(\frac{\alpha_{k-1}-\alpha_k}{\alpha_k}\right)^2 \frac{1}{\bar{\alpha}\bar{\gamma}k^{-(c_1+c_2)}}$$

for all such k. Substituting $\alpha_k = \bar{\alpha}k^{-c_1}$ and expanding the squared term gives

$$2\left(\frac{\alpha_{k-1}-\alpha_k}{\alpha_k}\right)^2 \frac{1}{\bar{\alpha}\bar{\gamma}k^{-(c_1+c_2)}} = 2\frac{\left[\left(1-\frac{1}{k}\right)^{-2c_1} - 2\left(1-\frac{1}{k}\right)^{-c_1}+1\right]}{\bar{\alpha}\bar{\gamma}k^{-(c_1+c_2)}}. \tag{5.15}$$

To approximate the terms containing $1/k$, we use a truncated power series expansion, namely that for $x \in (-1,1)$

$$(1-x)^{-r} \approx 1 + rx + \frac{1}{2}r(r+1)x^2 + \frac{1}{6}r(r+1)(r+2)x^3. \tag{5.16}$$

Applying Equation (5.16) to Equation (5.15) gives

$$2\frac{\left[\left(1-\frac{1}{k}\right)^{-2c_1} - 2\left(1-\frac{1}{k}\right)^{-c_1}+1\right]}{\bar{\alpha}\bar{\gamma}k^{-(c_1+c_2)}} \approx \frac{2c_1^2}{\bar{\alpha}\bar{\gamma}k^{2-(c_1+c_2)}} + \frac{2(c_1^3+c_1^2)}{\bar{\alpha}\bar{\gamma}k^{3-(c_1+c_2)}}.$$

We see that sums of such terms are given by

$$\sum_{k=1}^{\infty} 2\frac{\left[\left(1-\frac{1}{k}\right)^{-2c_1} - 2\left(1-\frac{1}{k}\right)^{-c_1}+1\right]}{\bar{\alpha}\bar{\gamma}k^{-(c_1+c_2)}} \approx \frac{2c_1^2}{\bar{\alpha}\bar{\gamma}}\zeta(2-(c_1+c_2)) + \frac{2(c_1^3+c_1^2)}{\bar{\alpha}\bar{\gamma}}\zeta(3-(c_1+c_2)). \tag{5.17}$$

Returning to Equation (5.13) and using the results of Equations (5.14) and (5.17) gives

$$\sum_{k=1}^{\infty}\sigma_k \lessapprox \bar{\gamma}^2 K_w \zeta(2c_2) + \frac{2c_1^2}{\bar{\alpha}\bar{\gamma}}\zeta(2-(c_1+c_2)) + \frac{2(c_1^3+c_1^2)}{\bar{\alpha}\bar{\gamma}}\zeta(3-(c_1+c_2)). \tag{5.18}$$

Due to the approximations made and ranges of k considered in bounding this sum, we can only guarantee that the convergence estimate relying on $\sum_{k=1}^{\infty}\sigma_k$ will hold for $k \geq \max\{M, \hat{M}\}$. However, for $k \leq M$ we will often have $\sigma_k < 0$ (as when L_G is large), and thus we expect the

bound in Equation (5.18) to hold for a range of values of $k \leq M$ because negative terms with such indices have been over-estimated by including positive terms at such indices in Equation (5.18). In addition, we expect $\bar{\alpha}$ and $\bar{\gamma}$ to be small enough that $\hat{M}$ will often be small, e.g., less than 10, thus allowing this bound to hold over a wide range of values of k.

To apply the bound in Equation (5.10), we also need to estimate the term $\mathbb{E}[\|z(k) - \xi_{k-1}\|^2]$. Returning to Equation (5.12) and taking the expectation of both sides one timestep earlier gives

$$\mathbb{E}[\|z(k) - \xi_{k-1}\|^2] \leq (1 - \tau_{k-1})\mathbb{E}[\|z(k-1) - \xi_{k-2}\|^2] + \sigma_{k-1}, \tag{5.19}$$

which is a time-varying affine recurrence relation in the expected error in the optimization algorithm. Solving Equation (5.19) (see e.g., [98], Section 2.1.1.2), we find that

$$\mathbb{E}[\|z(k) - \xi_{k-1}\|^2] = T(k)\big(\mathbb{E}[\|z(1) - \xi_0\|^2] + S(k)\big),$$

where

$$T(k) = \prod_{n=0}^{k-1}(1 - \tau_n)$$

and

$$S(k) = \sum_{m=0}^{k-1} \frac{\sigma_m}{\prod_{i=0}^{m}(1 - \tau_i)}.$$

Defining the diameter of the set Z via $D_z = \sup_{z_1, z_2 \in Z} \|z_1 - z_2\|$, we can bound the initial error via $\mathbb{E}[\|z(1) - \xi_0\|^2] \leq D_z^2$. With this bound, we state the following convergence theorem for Algorithm 5.2.

Theorem 5.4 *Let all hypotheses of Theorem 5.3 hold, suppose that* $\{\sigma_k\}_{k \in \mathbb{N}}$ *is summable, and*

suppose that Equation (5.18) *holds with* $k = 1$*. Then, for all* $\varepsilon > 0$*,*

$$\mathbb{P}\big(\|z(j) - \xi_{j-1}\|^2 \leq \varepsilon \text{ for all } j \geq k\big) \geq$$
$$1 - \frac{1}{\varepsilon}\left(T(k)\big(D_z^2 + S(k)\big) + \bar{\gamma}^2 K_w \zeta(2c_2) + \frac{2c_1^2}{\bar{\alpha}\bar{\gamma}}\zeta\big(2-(c_1+c_2)\big) + \frac{2(c_1^3+c_1^2)}{\bar{\alpha}\bar{\gamma}}\zeta\big(3-(c_1+c_2)\big) - \sum_{n=0}^{k}\sigma_n\right).$$

Proof: *This follows from Theorem 5.3, Equation* (5.18)*, and the fact that* $\mathbb{E}[\|z(k) - \xi_{k-1}\|^2] \leq D_z^2$*.*

■

One may of course wonder how the scale of a problem affects the convergence of Algorithm 5.2. This point can be assessed using Theorem 5.4. The expected error in Theorem 5.4 is affected by σ_i, which in turn depends upon L_G, the Lipschitz constant of the operator G. As one adds more constraints and per-agent objectives to a problem, certainly L_G becomes larger, though the specific choices of objectives and constraints can dramatically impact how L_G changes as a problem is scaled up. Thus the issue of scalability is largely dependent upon the functions used in a problem and their Lipschitz constants, with higher Lipschitz constants generally slowing convergence more. In addition, the choices of γ_k and α_k can help limit the changes in σ_k in response to growth in L_G, letting users improve scalability through judicious choices of parameters.

Having explored convergence in the presence of privacy, we now examine the trade-off between the two competing objectives of privacy and convergence.

5.3.3 The Trade-off Between Privacy and Convergence

In this section we derive a quantifiable trade-off between privacy and convergence, and for concreteness we focus on the case of ε-differential privacy, though a similar trade-off can be derived for (ε, δ)-differential privacy.

Returning to Equation (5.11) we find the inequality

$$\mathbb{E}[\|z(k+1) - \xi_k\|^2] \leq \gamma_k^2 \mathbb{E}[\|w(k)\|^2] + \theta_k(1 + \alpha_k\gamma_k)\|z(k) - \xi_{k-1}\|^2 + \theta_k\rho_k, \tag{5.20}$$

where we see that only the term $\mathbb{E}[\|w(k)\|^2]$ depends upon the noise added for privacy. Given that $w(k)$ has zero mean, we find $\mathbb{E}[\|w(k)\|^2] = \text{Var}\big[w(k)\big]$. In the case of ε-differential privacy, we have $w(k) \sim \text{Lap}(0, b/\varepsilon)^r$ so that

$$\text{Var}[w(k)] = \frac{W}{\varepsilon^2},$$

where $W := W(\Delta_1 g, \Delta_1 g_{x_i}, B)$ is a constant that depends upon the systems of interest, g and g_{x_i}, and the adjacency parameter, B. Returning to Equation (5.20) and substituting in $\text{Var}[w(k)] = \frac{W}{\varepsilon^2}$, we find

$$\mathbb{E}[\|z(k+1) - \xi_k\|^2] \leq \theta_k(1 + \alpha_k\gamma_k)\|z(k) - \xi_{k-1}\|^2 + \theta_k\rho_k + \gamma_k^2\frac{W}{\varepsilon^2}. \tag{5.21}$$

The additive term $\gamma_k^2\frac{W}{\varepsilon^2}$ is the only term in which the privacy parameter ε appears, and this term can be regarded as a penalty on convergence because it allows the expected error $\mathbb{E}[\|z(k+1) - \xi_k\|^2]$ to grow from $\|z(k) - \xi_{k-1}\|$. Viewing this term as a convergence penalty then reveals a fundamental trade-off between privacy and convergence: implementing ε-differential privacy comes at the cost of a convergence penalty proportional to $1/\varepsilon^2$. We state this trade-off succinctly and informally by writing

$$\text{Privacy}(\varepsilon) \iff \text{Convergence}\left(\frac{1}{\varepsilon^2}\right).$$

One can also see the effects of removing privacy from Algorithm 5.2 in Equation (5.21). There, eliminating privacy corresponds to $W = 0$, which in turn removes the term $\gamma_k^2\frac{W}{\varepsilon^2}$. Thus the introduction of privacy into Algorithm 5.1 introduces a convergence error of the form $\gamma_k^2\frac{W}{\varepsilon^2}$ at time k.

5.4 Simulation Results

Below we present numerical simulation results for a system with $n = 10$ agents and $m = 6$ constraints. We simulate both ε- and (ε, δ)-differential privacy.

5.4.1 Example Problem

Let there be $n = 10$ agents, each with state $x_i \in \mathbb{R}^2$ and using ensemble objective function

$$f(x) = \big((x_{1,1}-5)+(x_{1,2}+5)\big)+\|x_2\|^2+\left\|x_3 - \begin{pmatrix} -7 \\ 7 \end{pmatrix}\right\|^2+\big((x_{4,1}-8)+(x_{4,2}-8)\big)+\left\|x_5 + \begin{pmatrix} 3 \\ 3 \end{pmatrix}\right\|^4$$
$$+ \big((x_{6,1}-10)+(x_{6,2}-10)\big) + \big((x_{7,1}+10)+(x_{7,2}+10)\big)$$
$$+\left\|x_8 + \begin{pmatrix} 7 \\ 0 \end{pmatrix}\right\|^2 + \big((x_{9,1}-6)+x_{9,2}\big)+\left\|x_{10} - \begin{pmatrix} 0 \\ 8 \end{pmatrix}\right\|^4,$$

where $x_{i,j}$ is the j^{th} state of agent i and the per-agent objectives can be discerned in the obvious way. The constraints on the agents are

$$g(x) = \begin{pmatrix} \|x_1\|^2 + \|x_2\|^2 + \|x_3\|^2 - 10 \\ \|x_4\|^2 + \|x_5\|^2 + \|x_6\|^2 - 50 \\ \|x_7\|^2 + \|x_8\|^2 + \|x_9\|^2 - 50 \\ x_{1,1}^2 + x_{5,1} + x_{10,1}^2 - 50 \\ x_{4,2}^2 + x_{7,1} + x_{9,2} - 20 \\ \|x_8\|^2 + \|x_6\|^2 - 30 \end{pmatrix} \leq 0.$$

Each agent was also constrained to lie in the box $X_i = [-10, 10] \times [-10, 10]$. The Lipschitz constants of g were computed to be $K_1^g = 39.82$ and $K_2^g = 56.71$. The Lipschitz constants for each g_{x_i} are shown in Table 5.1.

In both simulation runs below, the step-size rule discussed at the end of Section 5.3 was used with the values

$$\bar{\alpha} = 0.1,\ \bar{\gamma} = 0.01,\ c_1 = 0.3,\ \text{and } c_2 = 0.52,$$

and all states and Kuhn-Tucker multipliers were initialized to zero, i.e., $x_i(0) = 0$ for all $i \in I$ and $\mu(0) = 0$.

Table 5.1: Values of the Lipschitz constants K_1^i and K_2^i for g_{x_i}, $i \in \{1, \ldots, 10\}$, used in implementing differential privacy in Algorithm 5.2.

i	K_1^i	K_2^i
1	4	$\sqrt{8}$
2	2	2
3	2	2
4	2	2
5	2	2
6	4	$\sqrt{8}$
7	2	2
8	4	$\sqrt{8}$
9	2	2
10	2	2

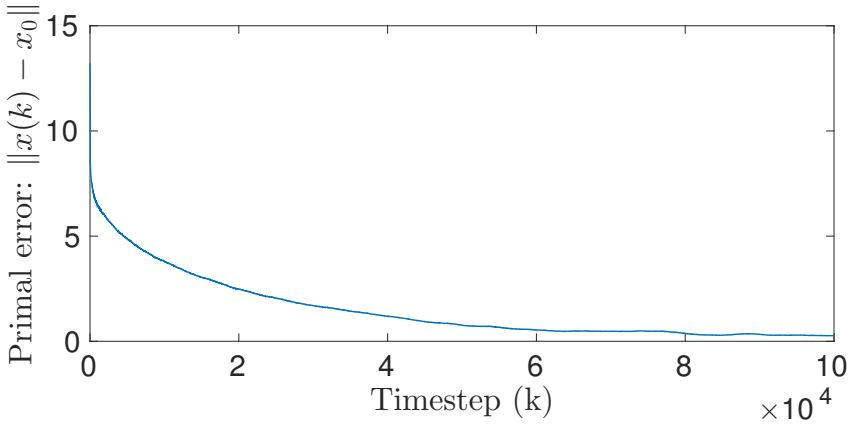

Figure 5.1: The values of $\|x(k) - x_0\|_2$ for $k = 1, \ldots, 100,000$ under ε-differential privacy with Algorithm 5.2. The steady, monotone descent toward x_0 indicates numerical convergence to x_0 in the presence of noise.

5.4.2 Simulation of ε-differential privacy

The adjacency parameter was chosen to be $B = 1$. The value $\varepsilon = \log 2$ was used for all systems. The distribution and variance of each entry of each noisy signal were computed and are listed in Table 5.2, where we use the notation for the PDF of a random scalar with the understanding that each entry of the random matrices w_i was generated using such a distribution.

Table 5.2: Noisy signals and their distributions for implementing ε-differential privacy in Algorithm 5.2.

Noise	Distribution	Variance
w_1	$\text{Lap}(0, 5.771)$	66.60
w_2	$\text{Lap}(0, 2.885)$	16.65
w_3	$\text{Lap}(0, 2.885)$	16.65
w_4	$\text{Lap}(0, 2.885)$	16.65
w_5	$\text{Lap}(0, 2.885)$	16.65
w_6	$\text{Lap}(0, 5.771)$	66.60
w_7	$\text{Lap}(0, 2.885)$	16.65
w_8	$\text{Lap}(0, 5.771)$	66.60
w_9	$\text{Lap}(0, 2.885)$	16.65
w_{10}	$\text{Lap}(0, 2.885)$	16.65

The distribution for w_g was $\text{Lap}(0, 57.45)$ with variance $6.600 \cdot 10^3$. Using this problem formulation, Algorithm 5.2 was run for $100,000$ iterations. To show the behavior of the algorithm over time, the least-norm saddle point of L, $z_0 = (x_0, \mu_0)$, was computed ahead of time and the values of $\|x(k) - x_0\|_2$ and $\|\mu(k) - \mu_0\|_2$ are shown in Figures 5.1 and 5.2, respectively, for $1 \leq k \leq 100,000$[2].

In Figures 5.1 and 5.2 we see a clear decreasing trend in both $\|x(k) - x_0\|_2$ and $\|\mu(k) - \mu_0\|_2$, with the primal error appearing to be monotonically decreasing and the dual error oscillating while showing a general decreasing trend. The oscillations seen are expected given that the variance of the noises added is constant while $\|G\|$ decreases in magnitude as the saddle point z_0 is approached. In fact, it is known that descent will be achieved in a gradient method as long as the norm of noise added to the gradient is less than the norm of the gradient itself [99]. In light of this fact, the trends

[2]Though the 1-norm is used for other aspects of ε-differential privacy, we measure distance to z_0 using the 2-norm to allow for meaningful visual comparison of the plots corresponding to ε-differential privacy in this subsection to those corresponding to (ε, δ)-differential privacy in the next subsection.

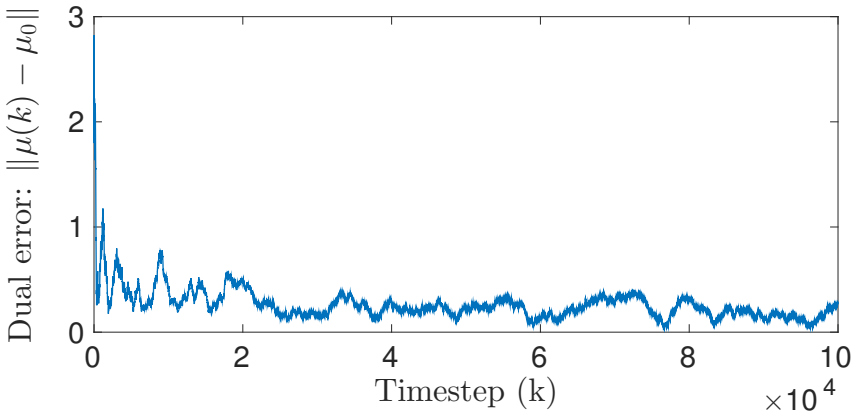

Figure 5.2: The values of $\|\mu(k) - \mu_0\|_2$ for $k = 1, \ldots, 100,000$ under ε-differential privacy with Algorithm 5.2. Here we see an initial descent followed by a period of oscillations as $\mu(k)$ approaches μ_0.

seen in Figures 5.1 and 5.2 are not surprising because the gradients in G in Algorithm 5.2 will have large norms far from z_0, thereby allowing them to "overpower" the noise added, while close to z_0 their norms will be smaller and the noise can dominate, causing increases in the distance to z_0 at some timesteps. Of course, $z(k) \to z_0$ asymptotically because these increases in $\|z(k) - z_0\|_2$ average out over very long periods of time.

The initial error values here were

$$\|x(0) - x_0\|_2 = 13.19 \text{ and } \|\mu(0) - \mu_0\|_2 = 2.169,$$

And in this run the final error values were

$$\|x(100,000) - x_0\|_2 = 0.2706 \text{ and } \|\mu(100,000) - \mu_0\|_2 = 0.2842,$$

with these values after half of the total runtime being

$$\|x(50,000) - x_0\|_2 = 0.7658 \text{ and } \|\mu(50,000) - \mu_0\|_2 = 0.2225.$$

These values confirm what can be seen visually in Figures 5.1 and 5.2: shorter runtimes than

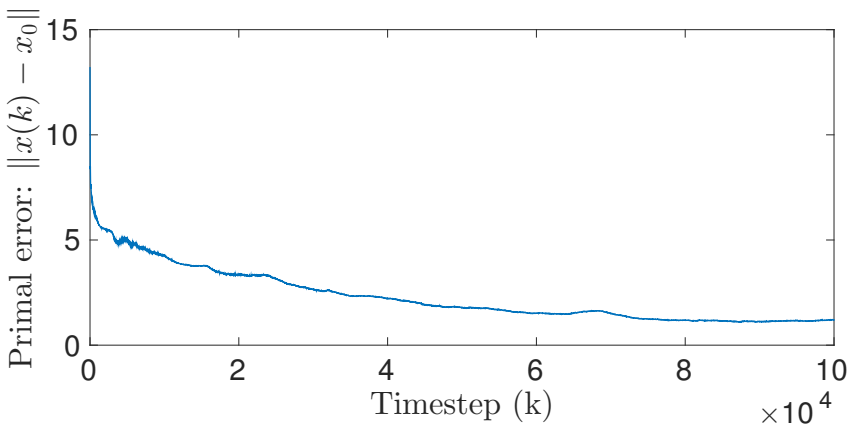

Figure 5.3: The values of $\|x(k) - x_0\|_2$ for $k = 1, \ldots, 100,000$ under (ε, δ)-differential privacy with Algorithm 5.2. The rapid descent toward x_0 and clear decreasing trend thereafter indicate numerical convergence to x_0 in the presence of noise.

$100,000$ timesteps can be used while ending at a reasonable distance from z_0 and, in light of the large variances of some noises present, reasonable numbers of iterations produce an approach toward z_0 that would be useful in many applications.

5.4.3 Simulation of (ε, δ)-differential privacy

In this case the adjacency parameter was chosen to be $B = 1$. The values $\varepsilon = \log 2$ and $\delta = 0.01$ were used for all systems, giving $\kappa(\delta, \varepsilon) = 3.559$. Using this privacy policy, the distribution and variance of each noisy signal were computed and are listed in Table 5.3.

The distribution for w_g was $\mathcal{N}(0_{6\times 1}, 4.073 \cdot 10^4 I_{6\times 6})$ with variance $4.073 \cdot 10^4$. In Table 5.3 we record the distribution of each entry of the matrices w_i, $i \in I$, with the understanding that each w_i has i.i.d. entries. Using this problem formulation, Algorithm 5.2 was run for $100,000$ iterations and the values of $\|x(k) - x_0\|_2$ and $\|\mu(k) - \mu_0\|_2$ for $1 \leq k \leq 100,000$ are plotted in Figures 5.3 and 5.4, respectively. In Figures 5.3 and 5.4 we see a similar trend to Figures 5.1 and 5.2: nearly monotone decreases in the primal error, and general decreases in the dual error with noticeable oscillations present.

Table 5.3: Noisy signals and their distributions for implementing (ε, δ)-differential privacy in Algorithm 5.2.

Noise	Distribution	Variance
w_1	$\mathcal{N}(0, 101.3)$	101.3
w_2	$\mathcal{N}(0, 50.66)$	50.66
w_3	$\mathcal{N}(0, 50.66)$	50.66
w_4	$\mathcal{N}(0, 50.66)$	50.66
w_5	$\mathcal{N}(0, 50.66)$	50.66
w_6	$\mathcal{N}(0, 101.3)$	101.3
w_7	$\mathcal{N}(0, 50.66)$	50.66
w_8	$\mathcal{N}(0, 101.3)$	101.3
w_9	$\mathcal{N}(0, 50.66)$	50.66
w_{10}	$\mathcal{N}(0, 50.66)$	50.66

The initial error values for this run were

$$\|x(0) - x_0\|_2 = 13.19 \text{ and } \|\mu(0) - \mu_0\|_2 = 2.169.$$

The final error values here were

$$\|x(100,000) - x_0\|_2 = 1.1965 \text{ and } \|\mu(100,000) - \mu_0\|_2 = 0.7413,$$

while after half of the total timesteps taken these values were

$$\|x(50,000) - x_0\|_2 = 1.7857 \text{ and } \|\mu(50,000) - \mu_0\|_2 = 0.2500,$$

indicating a rapid initial descent towards z_0 and close proximity to it thereafter.

Both simulation examples show a rapid decrease in the distance from $z(k)$ to z_0. Such a rapid decrease lends itself to use of this algorithm in practical applications because it allows for useful improvements to be made in the value of f in a reasonable runtime while respecting the set and functional constraints of the problem. The theoretical and simulation results presented here demonstrate the utility of the iterative Tikhonov regularization, even in the presence of noise with

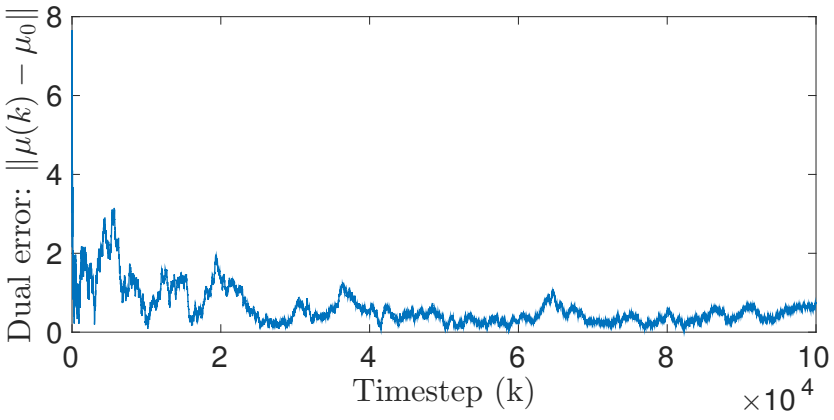

Figure 5.4: The values of $\|\mu(k) - \mu_0\|_2$ for $k = 1, \ldots, 100,000$ under (ε, δ)-differential privacy with Algorithm 5.2. The initial approach toward μ_0 and oscillations in distance beyond that point indicate numerical convergence to μ_0 when noise is added for differential privacy.

large variance. This robustness is further supported by the simulation results in [32] and demonstrates that, in a practical setting, strong, quantifiable guarantees of privacy can be achieved while providing useful convergence guarantees in the optimization problem of interest. Critical to the success of these numerical results is all noise being zero mean, and it is a feature of differential privacy that zero mean noise is effective at protecting sensitive information.

5.5 Conclusion

A differentially private optimization algorithm for teams of many agents coordinated by a central cloud computer was presented. This problem was treated as a stochastic variational inequality and solved using a Tikhonov-regularized Goldstein-Levitin-Polyak iteration. Its convergence was shown for both ε- and (ε, δ)-differential privacy, and numerical convergence of the algorithm was shown in simulation, demonstrating the ability to arrive at a collective decision while maintaining privacy for the users involved in making it.

It was further shown that there is a quantifiable trade-off between privacy and convergence, and this trade-off provides a quantitative tool that network operators can use to rigorously balance the two competing goals of user privacy and network coordination. A key feature of this work is that

it implements privacy seamlessly, in the sense that anything that can be done without privacy can be done with privacy, and the same solution will be reached in a strong probabilistic sense.

CHAPTER 6

APPROXIMATELY-TRUTHFUL MULTI-AGENT COORDINATION VIA JOINT DIFFERENTIAL PRIVACY

Multi-agent optimization problems have found applications in a range of research areas, including power systems in [65], machine learning in [26], sensor networks in [3], and robotics in [6]. Solutions to some problems in these applications rely on the implicit assumption that all agents share correct, truthful information with others in a network. However, one can envision a scenario in which deceitful agents do not honestly share their data, instead sharing false information that will skew the behavior of the system in their favor. For example, a homeowner connected to a smart power grid may report a false value for his or her power usage in order to save money. This chapter considers optimization problems with agents that may be intentionally deceitful for their own benefit, and it provides a method for disincentivizing untruthful behavior when teams of agents are collectively optimizing.

We reduce the incentive to share false information by using *joint differential privacy*, defined in [100], to limit the possible decrease in cost an agent can achieve through intentionally misreporting its state. Joint differential privacy adds noise to reduce the ability of an agent to benefit from reporting false information, and was first introduced for this purpose in [100] to promote truthful sharing of information by the players in a class of games. This work was applied specifically to optimization problems arising from distributed electric vehicle charging in [101], as well as to linearly separable optimization problems in [102].

The developments in this chapter differ from those in [101] and [102], as well as some other work on private optimization, e.g., [103], in two key ways. First, the work in [101], [102], and [103] uses differential privacy as it was originally defined in [79] to keep some static object, such as a constraint function, private. Our work focuses on trajectory-level privacy for state trajectories

that are dynamically generated as an optimization algorithm is executed. In applications such as phase synchronization in smart power grids, each iterate of an optimization algorithm corresponds to a physical state at some point in time, and an agent's contribution to the optimization process is its whole state trajectory. Applying joint differential privacy in such applications should therefore be done at the trajectory level, and implementing joint differential privacy for trajectory-level data is most naturally done using the dynamical systems formulation of differential privacy. Therefore, this chapter uses trajectory-level privacy as defined in [82], rather than privacy for databases as in [79].

This form of privacy is not only better suited to the applications of interest, but also allows privacy guarantees to hold across infinite time horizons which, in many cases, cannot be attained using privacy for databases. To our knowledge, this work is the first use of joint differential privacy at the trajectory level. The second key difference between the existing literature and this work is the class of optimization problems solved. We apply joint differential privacy in a general multi-agent nonlinear programming setting, incorporating non-separable functional constraints that are also possibly nonlinear, which differs from work done in [101], [102], and [103] where either linear or affine constraints are considered.

To solve such problems, a cloud computer is added to the team of agents to serve as a trusted central aggregator, also sometimes called a "curator" in the privacy literature. This architecture was previously used for privately solving multi-agent nonlinear programs in Chapter 5, in which honest agents seek to protect sensitive data from eavesdroppers via ordinary differential privacy. The work in [104] suggests that any differential privacy implementation provides some disincentive against untruthful information sharing, and the current work uses the notion of joint differential privacy to formally provide this guarantee in the framework developed in Chapter 5. Accordingly, the technical novelty of this chapter is not in the algorithm used to solve problems, but in the theoretical performance guarantees that are provided using this framework; the work in Chapter 5 focuses exclusively on protecting sensitive data, but the current chapter focuses on the problem of preventing untruthful behavior by the agents. The technical contribution of this chapter thus con-

sists of adapting our existing privacy framework to the problem of incentivizing truthful behavior by the agents, and quantifying the extent to which any agent can benefit from untruthful behavior.

Several existing approaches use behavioral analysis to identify untruthful agents, including those in [105] and [106]. In this chapter, an agent's local state updates rely on a local objective function and local constraint set that are considered sensitive, and therefore these local data are not shared with any other agent. As a result, the correct next value of an agent's state is only known to that agent, and a behavioral analysis approach cannot be used here because no outside observer can determine what any agent's future states should be. Rather than detecting untruthful behavior, this chapter seeks to prevent it outright by reducing the incentive to share untruthful information via joint differential privacy. The principle underlying this application of joint differential privacy is that adding noise to the system can make each agent's cost insensitive to changes in the agent's state trajectory. The amount of noise added must be calibrated to the system in order to dilute the effect of an agent reporting an untruthful state to the cloud, and this chapter presents this calibration for a general multi-agent nonlinear program in terms of constants pertaining to the problem.

The problems considered consist of a collection of agents, each with a local objective function and local set constraint, and ensemble state constraints that jointly constrain the agents. A primal-dual approach is used, in which the agents update their own states, which are the problem's primal variables, and the cloud updates the problem's dual variables. Naturally, the constraints in this problem will usually lead to higher costs for each agent than a comparable unconstrained problem. As a result, some of the agents may wish to skew the constraints in their favor. One way they may do so is by reporting false state information to the cloud in order to loosen the constraints' effects on their on their own states, thereby giving the untruthful agents a lower cost. This manipulation of the constraints will result in an unequal distribution of the burden of these constraints by tightening them on honest agents. Therefore, this form of untruthful behavior is disincentivized using joint differential privacy.

The rest of the chapter is organized as follows. Section 6.1 provides the necessary optimization background, the structure of communications in the system, and formally states the joint differen-

tially private optimization problem that is the focus of the chapter. Then, Section 6.2 reviews joint differential privacy, and Section 6.3 presents the proposed joint differentially private algorithm. Next, Section 6.4 proves the main result of the chapter on limiting an agent's incentive to misreport its states to the cloud. Section 6.5 then presents simulation results, and Section 6.6 provides concluding remarks.

6.1 Background and Problem Statement

This section presents the multi-agent optimization problem of interest and an optimization algorithm that will later be used to solve a joint differentially private version of it. This section also describes the cloud-based architecture used in the remainder of the chapter. Throughout the chapter, ∇_{x_i} denotes the partial derivative with respect to x_i, and h_{x_i} denotes the partial derivative of the function h with respect to x_i.

6.1.1 Optimization Problem Formulation

Consider a problem consisting of N agents indexed over $i \in [N] := \{1, \dots, N\}$. Agent i has state $x_i \in \mathbb{R}^{n_i}$ with $n_i \in \mathbb{N}$ and a local set constraint of the form $x_i \in X_i \subseteq \mathbb{R}^{n_i}$. The diameter of X_i is denoted by $D_i := \sup_{x_1, x_2 \in X_i} \|x_1 - x_2\|_1$. Agent i also has a local objective function $f_i : \mathbb{R}^{n_i} \to \mathbb{R}$ depending only upon its own state. Agent i's local data are subject to the following assumption.

Assumption 6.1 *For all $i \in [N]$, f_i is C^2 and convex in x_i, and X_i is non-empty, compact, and convex.* $\lozenge$

Assumption 6.1 implies that f_i is Lipschitz and its Lipschitz[1] constant is denoted by K_i. In particular, Assumption 6.1 allows for convex polynomial objectives and box constraints, which are common in multi-agent optimization problems. Both f_i and X_i are considered sensitive information and are therefore not shared with any other agents or with the cloud. For simplicity of notation, define the set $X := X_1 \times \cdots \times X_N \subseteq \mathbb{R}^n$, where $n = \sum_{i \in [N]} n_i$. The agents' individual

[1] All Lipschitz constants in this chapter are with respect to the metric induced by the 1-norm.

set constraints require $x \in X$, where x is the ensemble state vector of the network, defined as $x = \left(x_1^T, \ldots x_N^T\right)^T \in X$, and where Assumption 6.1 provides that X is non-empty, compact, and convex.

The agents are together subject to global inequality constraints $g(x) \leq 0$, where $g : \mathbb{R}^n \to \mathbb{R}^m$. The functional constraints in g are subject to the following assumption.

Assumption 6.2

1. *For all $j \in [m]$, the constraint g_j is C^2 and convex in x.*

2. *There exists a point $\bar{x} \in X$ such that $g(\bar{x}) < 0$.*

$\lozenge$

Assumption 6.2.1 admits a wide variety of constraint functions, e.g., any convex polynomials. Assumption 6.2.2 is known as Slater's condition, e.g., Assumption 6.4.2 in [28], and, in conjunction with Assumption 6.2.1, guarantees that strong duality holds. Assumption 6.2 implies that g and each g_{x_i} are Lipschitz. Their Lipschitz constants are denoted by K_g and $L_{g,i}$, respectively.

Summing the per-agent objective functions gives the ensemble objective $f(x) = \sum_{i \in [N]} f_i(x_i)$, which is C^2 and convex in x because of Assumption 6.1. Together f, g, and X comprise the following ensemble-level optimization problem.

Problem 6.0 *(Preliminary; no joint differential privacy yet)*

$$\begin{aligned} &\text{minimize } f(x) \\ &\text{subject to } g(x) \leq 0 \\ &\qquad x \in X. \end{aligned}$$

$\blacklozenge$

We now detail how to solve Problem 6.0 in the cloud-based system, and then give a unified problem statement, including incentivizing truthful behavior, in Problem 6.1 below.

The Lagrangian associated with Problem 6.0 is

$$L(x, \mu) = f(x) + \mu^T g(x),$$

where $\mu \in \mathbb{R}^m_+$ is the dual vector associated with Problem 6.0 and $\mathbb{R}^m_+$ denotes the non-negative orthant of $\mathbb{R}^m$. Seminal work in [29] shows that a point $\hat{x} \in X$ is a solution to Problem 6.0 if and only if there exists a point $\hat{\mu} \in \mathbb{R}^m_+$ such that $(\hat{x}, \hat{\mu})$ is a saddle point of L. This saddle point condition can be compactly expressed by requiring

$$L(\hat{x}, \mu) \leq L(\hat{x}, \hat{\mu}) \leq L(x, \hat{\mu}) \text{ for all } (x, \mu) \in X \times \mathbb{R}^m_+, \tag{6.1}$$

and an optimal primal-dual pair $(\hat{x}, \hat{\mu})$ exists under Assumptions 6.1 and 6.2.

For the forthcoming optimization algorithm, Equation (6.1) is used to define an upper bound on the norm of $\hat{\mu}$, stated in the following lemma. It uses the Slater point $\bar{x}$ from Assumption 6.2.2 and any lower bound on f over X, denoted f_{lower}, which exists under Assumption 6.1.

Lemma 6.1 *For $(\hat{x}, \hat{\mu})$ a saddle point of L,*

$$\hat{\mu} \in M := \left\{ \mu \in \mathbb{R}^m_+ : \|\mu\|_1 \leq \mu_{max} := \frac{f(\bar{x}) - f_{lower}}{\min\limits_{j \in [m]} \{-g_j(\bar{x})\}} \right\}.$$

Proof: See Section II-A in [107]. ■

Using the fact that a saddle point of L provides a solution to Problem 6.0, the remainder of the chapter focuses on finding such saddle points. The algorithm used for this purpose includes an iterative Tikhonov regularization with an asymptotically vanishing stepsize, and was stated in [34] for deterministic variational inequalities and later in [91] for stochastic problems. To help describe the communications and computations in the cloud-based system, we provide the general deterministic form of this algorithm now, though in Section 6.3 stochasticity is introduced when the algorithm is made joint differentially private. The saddle-point finding algorithm uses the coupled

update equations

$$x(k+1) = \Pi_X \left[x(k) - \gamma_k \left(L_x(k) + \alpha_k x(k) \right) \right] \tag{6.2a}$$

$$\mu(k+1) = \Pi_M \left[\mu(k) + \gamma_k \left(L_\mu(k) - \alpha_k \mu(k) \right) \right], \tag{6.2b}$$

where Π_X and Π_M are the Euclidean projections onto X and M, respectively. This update law will be referred to as Update (6.2). Here, γ_k is a stepsize and α_k is the regularization parameter, and the values of both will be provided in Theorem 6.1 in Section 6.3.

6.1.2 Optimizing over the Cloud

We now elaborate on the cloud-based architecture to be used and the means of executing Update (6.2) across the cloud and agents. To enforce joint differential privacy, the agents do not directly share any information with each other. Instead, the agents route messages through a trusted cloud computer which aggregates all states in the network, performs computations involving these states, and makes the results of these computations joint differentially private before sending them to the agents.

Splitting Equation (6.2a) into each agent's state gives the update

$$x_i(k+1) = \Pi_{X_i} \left[x_i(k) - \gamma_k \left(L_{x_i}(k) + \alpha_k x_i(k) \right) \right]$$

for agent i, where expanding the term containing L_{x_i} gives

$$x_i(k+1) = \Pi_{X_i} \Big[x_i(k) - \gamma_k \Big(\nabla_{x_i} f_i\big(x_i(k)\big) + g_{x_i}\big(x(k)\big)^T \mu(k) + \alpha_k x_i(k) \Big) \Big]. \tag{6.3}$$

Importantly, the right-hand side of Equation (6.3) contains two terms which agent i cannot compute on its own: $g_{x_i}\big(x(k)\big)$, because it is a function of every state in the network, and $\mu(k)$, because its update law relies on $g\big(x(k-1)\big)$, which is also a function of every state in the network. For this reason, the cloud computer is used to compute $\mu(k)$ and $g_{x_i}\big(x(k)\big)$ for each $i \in [N]$ at all timesteps

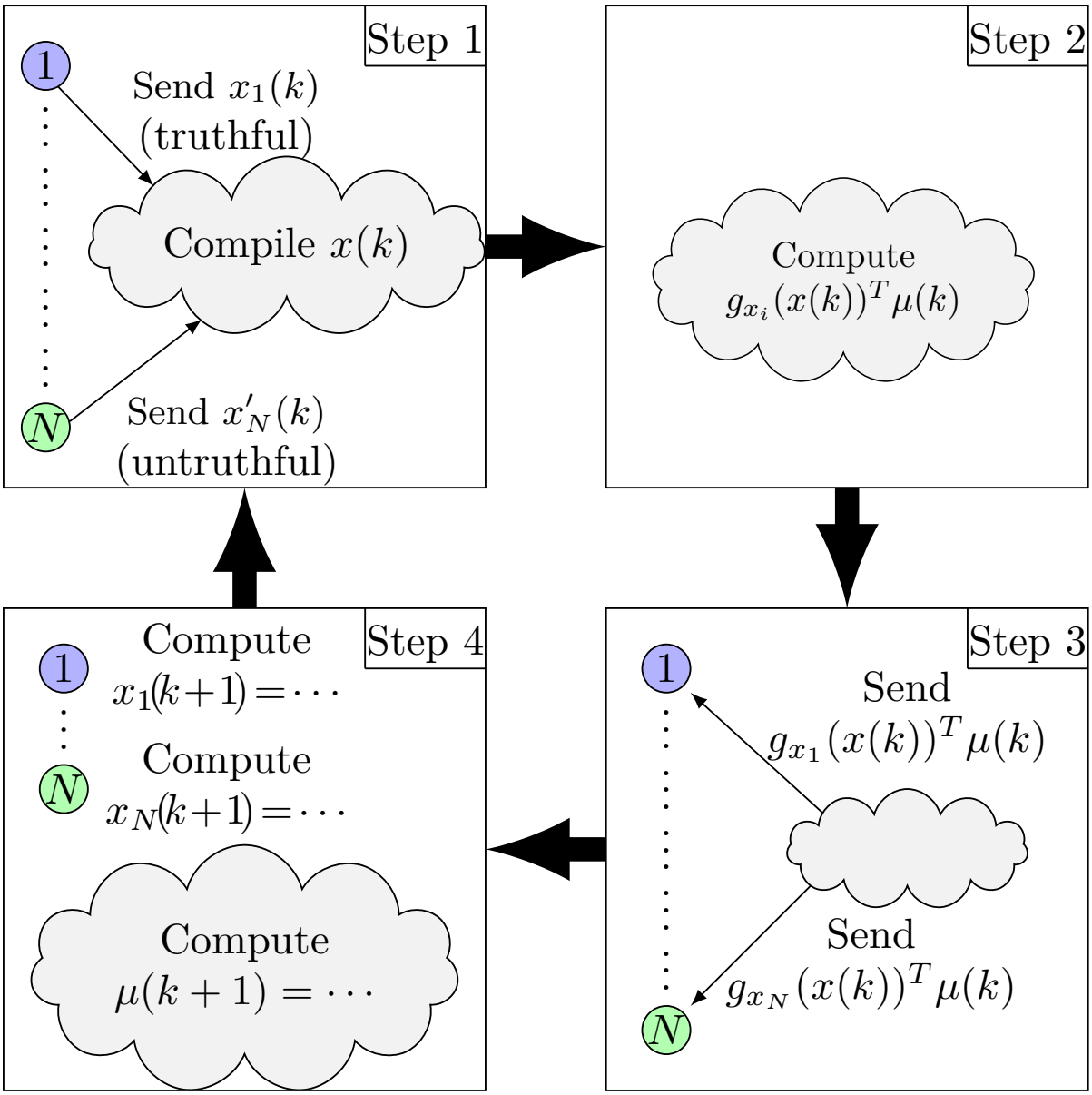

Figure 6.1: The four steps of a communications cycle in the cloud-based system. First, each agent sends its state to the cloud. Second, the cloud performs centralized computations required by the agents. Third, the cloud sends the results of these computations to the agents. Fourth, agent i computes $x_i(k+1)$ and the cloud computes $\mu(k+1)$, and then this process repeats. In Step 1 depicted here, agent N is misreporting its state to the cloud by sending $x'_N(k)$ instead of $x_N(k)$. As the algorithm progresses, this untruthful state propagates through the system.

k.

Four actions occur within timestep k. First, agent i sends $x_i(k)$ to the cloud and the cloud assembles the ensemble state vector $x(k)$. Second, the cloud computes $g_{x_i}\big(x(k)\big)$ for every $i \in [N]$. Third, the cloud sends $g_{x_i}\big(x(k)\big)^T \mu(k)$ to agent i. Fourth, the cloud computes $\mu(k+1)$ and simultaneously agent i computes $x_i(k+1)$, and then this process repeats. This exchange of information is depicted in Figure 6.1. In the upper-left panel of Figure 6.1, agents 1 through $N-1$ report their actual state to the cloud, while agent N misreports its state to the cloud by sending some $x'_N(k)$ instead of $x_N(k)$. The arrows connecting boxes indicate how misreported states propagate through the system, eventually affecting all agents' states. The cloud's computations in Steps 2 and 4 will be modified in Section 6.2 to introduce joint differential privacy, though the overall communications structure will remain the same.

To reflect that agent i receives the vector $g_{x_i}\big(x(k)\big)^T \mu(k)$ from the cloud without knowing $g_{x_i}\big(x(k)\big)$ or $\mu(k)$ individually, agent i's update law is rewritten as

$$x_i(k+1) = \Pi_{X_i}\Big[x_i(k) - \gamma_k \left(\nabla_i f_i\big(x_i(k)\big) + q_i(k) + \alpha_k x_i(k)\right)\Big],$$

where $\mathbb{R}^{n_i} \ni q_i(k) := g_{x_i}\big(x(k)\big)^T \mu(k)$. As in Equation (6.2b), the cloud computes the dual update according to

$$\mu(k+1) = \Pi_M \left[\mu(k) + \gamma_k \left(L_\mu(k) - \alpha_k \mu(k)\right)\right].$$

6.1.3 Full Problem Statement

We now state the algorithm that will be used throughout the remainder of the chapter. Below that, we identify the potential for misreporting states in this algorithm and state the problem that is later solved using joint differential privacy.

Algorithm 6.1

Step 0: *For all $i \in [N]$, initialize agent i with $x_i(0)$, X_i, f_i, $\{\alpha_k\}_{k\in\mathbb{N}}$, and $\{\gamma_k\}_{k\in\mathbb{N}}$. Initialize the cloud with $\bar{x}$, g, f_{lower}, $\{\alpha_k\}_{k\in\mathbb{N}}$, and $\{\gamma_k\}_{k\in\mathbb{N}}$. Let the cloud compute M before the system begins*

optimizing. Set $k = 0$.

Step 1: *For all* $i \in [N]$*, the cloud computes* $q_i(k)$ *and sends it to agent* i.

Step 2: *Agent* i *computes*

$$x_i(k+1) = \Pi_{X_i}\Big[x_i(k) - \gamma_k \left(\nabla_i f_i\big(x_i(k)\big) + q_i(k) + \alpha_k x_i(k)\right)\Big],$$

and sends the state value $x_i(k+1)$ *to the cloud.*

Step 3: *The cloud computes*

$$\mu(k+1) = \Pi_M \left[\mu(k) + \gamma_k \left(L_\mu(k) - \alpha_k \mu(k)\right)\right].$$

Step 4: *Set* $k := k+1$ *and return to Step 1.* △

In Step 2 of Algorithm 6.1, agent i may report a false state value to the cloud, sending some $\tilde{x}_i(k+1) \neq x_i(k+1)$. This is the misreporting behavior that this chapter seeks to prevent using the cloud. The only influence the cloud has upon agent i is through $q_i(k)$, and therefore the cloud must compute $q_i(k)$ in a way that incentivizes agent i to honestly report its state. The incentivization of this behavior is stated as Problem 6.1 below, and the remainder of the chapter focuses on solving Problem 6.1.

Problem 6.1 *Execute Algorithm 6.1 with the cloud computing* $q_i(k)$ *in a way that incentivizes agent* i *to honestly report* $x_i(k+1)$ *in Step* 2*, while still converging to a minimum.* ♦

It is assumed that each agent ultimately wants the constraints g to be satisfied, as would be the case when g corresponds to some mission-critical conditions that must be satisfied by the agents. However, an agent may wish to reduce the impact g has upon its own state in order to reduce its cost. One way of affecting g for this purpose is by reporting false states to the cloud over time; because all agents want g to be satisfied, a misreporting agent will still use $q_i(k)$ from the cloud in its state updates, but an agent can substantially influence these messages for its own benefit through misreport. In response to misreported states, other agents' messages from the cloud in

Update (6.2) will be affected in a way that compensates for agent i's false reported states, thereby resulting in an unfair distribution of the burden of g.

This behavior cannot be detected by the other agents or the cloud because only agent i knows f_i and X_i, making no other entity in the network capable of determining what agent i's state should be (cf. Equation (6.2a)). Therefore, rather than detecting manipulation of g, we seek to prevent this behavior. Joint differential privacy provides a framework for incentivizing truthful sharing of information and it is used here to prevent the agents from manipulating g.

Because agents are opportunistic but not malicious, they may send untruthful states to the cloud, but they will not send states that harm the system or prevent convergence of Algorithm 6.1. As a result, a misreported state trajectory will have some relationship to an agent's true state trajectory and this relationship is used in defining adjacency of signals in our joint differential privacy implementation. The next section details the manner in which noise is added to $q_i(k)$ using joint differential privacy, and Section 6.3 shows that this noise still allows for Algorithm 6.1 to reach a minimum.

6.2 Joint Differential Privacy

This section recalls necessary details from both ordinary differential privacy and joint differential privacy. Then it presents the joint differential privacy mechanism that will be implemented on the preceding cloud architecture. It is critically important to note that while this section discusses privacy, the ultimate goal is to apply joint differential privacy to induce truthful sharing of states. All of the content on privacy in this section should therefore be understood as making progress toward inducing truthful behavior.

6.2.1 Differential Privacy Background

Differential privacy as described in [79] was originally designed to keep individual database entries private whenever a database is queried, and this is done by adding noise to the responses to such queries. This idea was extended to dynamical systems in [82] in order to keep inputs to a system

private from anyone observing the outputs of that system. It is the dynamical systems notion of differential privacy that is used below.

The key idea behind differential privacy is that noise is added to make "adjacent" inputs produce "similar" outputs, and these notions are made rigorous below. As above, let there be N users, with the i^{th} user contributing a signal $u_i \in \tilde{\ell}_{p_i}^{s_i}$. The space $\tilde{\ell}_{p_i}^{s_i}$ is the space of sequences of s_i-vectors in which every finite truncation of every element has finite p_i-norm. More explicitly, with $u_i(k) \in \mathbb{R}^{s_i}$ denoting the k^{th} element of the signal u_i, define the truncation operator P_t according to

$$P_t u = \begin{cases} u(k) & k \leq t \\ 0 & k > t \end{cases}.$$

Then $u_i \in \tilde{\ell}_{p_i}^{s_i}$ if and only if $P_t u_i \in \ell_{p_i}^{s_i}$ for all $t \in \mathbb{N}$. The full input space is then defined by the Cartesian product $\tilde{\ell}_p^s = \prod_{i=1}^n \tilde{\ell}_{p_i}^{s_i}$. This chapter focuses on the case where $p_i = 1$ for all $i \in [N]$.

To formalize the notion of adjacency of inputs in $\tilde{\ell}_p^s$, we define a binary, symmetric adjacency relation Adj_B^i, which is parameterized by $B > 0$ and an index $i \in [N]$. Its definition uses the notation $u_{-i} = (u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_N)$. The adjacency relation takes the form

$$\text{Adj}_B^i : \tilde{\ell}_p^s \times \tilde{\ell}_p^s \to \{0, 1\}$$

and has the following definition, as stated in [82].

Definition 6.1 *Two inputs u, $\tilde{u} \in \tilde{\ell}_p^s$ satisfy $\text{Adj}_B^i(u, \tilde{u}) = 1$ for some $i \in [N]$ if and only if*

$$\|u_i - \tilde{u}_i\|_{p_i} \leq B \text{ and } u_{-i} = \tilde{u}_{-i}.$$

The signals u and $\tilde{u}$ are then said to be adjacent with respect to agent i. If i is arbitrary, the relation Adj_B is used, and u and $\tilde{u}$ are simply called adjacent. $\Diamond$

This section maintains use of the symbol u for system inputs (rather than x as will be in subsequent sections) to maintain continuity with the references cited here for private dynamical systems.

Inputs from $\tilde{\ell}_p^s$ are assumed to pass into a causal, deterministic system $\mathcal{G}$, which produces outputs in $\tilde{\ell}_q^r$. To define when two outputs are "similar," the notion of a *mechanism* is used. In the context of private dynamical systems, a mechanism is a means of adding noise to an otherwise deterministic system to provide privacy to that system's inputs. Formally, for a fixed a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, a mechanism is a map of the form

$$M : \tilde{\ell}_p^s \times \Omega \to \tilde{\ell}_q^r.$$

The mechanism M must provide differential privacy to its input trajectories, which fundamentally means that the output of M should be insensitive to changes in its inputs. Differential privacy captures this notion by requiring that, whenever $\text{Adj}_B(u, \tilde{u})$ holds, the probability distributions of Mu and $M\tilde{u}$ satisfy

$$\mathbb{P}(Mu \in S) \leq e^{\varepsilon}\mathbb{P}(M\tilde{u} \in S)$$

for all S in an appropriate σ-algebra.

Using a result from the literature, we now state a finite-time criterion which holds if and only if M keeps entire trajectories private. Below, the notation $v_{0:k} := \big(v(0), \ldots, v(k)\big)$ is used to refer to the first $k+1$ entries of $v \in \tilde{\ell}_q^r$.

Lemma 6.2 *Let $\varepsilon \geq 0$ be given. For a dynamical system, a mechanism M is ε-differentially private if and only if, for all u, $\tilde{u}$ satisfying $\text{Adj}_B(u, \tilde{u}) = 1$ and for all times k,*

$$\mathbb{P}\big((M(u))_{0:k} \in A\big) \leq e^{\varepsilon}\mathbb{P}\big((M(\tilde{u}))_{0:k} \in A\big) \text{ for all } A \in \mathcal{B}^{(k+1)r},$$

where $\mathcal{B}^d$ is the Borel sigma-algebra on $\mathbb{R}^d$ and r is the dimension of the output space.

Proof: See Lemma 2 in [82]. ■

In Lemma 6.2, the value of ε determines the level of privacy afforded to the input signals, and decreasing its value leads to improved privacy at the cost of adding higher variance noise. Typical values of ε in the literature range from 0.1 to $\log 3$.

6.2.2 Joint Differential Privacy

We now elaborate on the application of joint differential privacy to Problem 6.1. To promote truth-telling behaviors, limits are imposed on the ability of any agent to reduce its cost by reporting a false state trajectory to the cloud. These limits are enforced using joint differential privacy, which is a relaxation of ordinary differential privacy for use in multi-agent systems, and it will be shown that this framework is sufficient for the goal of reducing an agent's ability to benefit from misreporting its state. For joint differential privacy, the "system" of interest is comprised by the computations the cloud carries out in accordance with Update (6.2), and this point is discussed further below. For now it suffices to point out that the output of this system is a tuple of private forms of all q_i's. Denoting the private form of $q_i(k)$ by $\tilde{q}_i(k)$ (whose exact form will be given later), the output of the cloud at time k is denoted by

$$y(k) = \big(\tilde{q}_1(k), \ldots, \tilde{q}_N(k)\big).$$

Let $\mathcal{M}$ denote a mechanism for joint differential privacy and let $\mathcal{M}_{-i}$ denote the same mechanism with the i^{th} output removed, i.e., the output of $\mathcal{M}_{-i}$ is

$$y_{-i}(k) = \big(\tilde{q}_1(k), \ldots, \tilde{q}_{i-1}(k), \tilde{q}_{i+1}(k), \ldots, \tilde{q}_N(k)\big).$$

A mechanism $\mathcal{M}$ is joint differentially private if, for any $i \in [N]$, $\mathcal{M}_{-i}$ preserves differential privacy for inputs adjacent with respect to i. Joint differential privacy for databases has been defined in [100], though, to our knowledge, it has not yet been used for dynamical systems. Using Lemma 2 in [82], the following lemma states a finite-time criterion for trajectory-level joint differential privacy.

Lemma 6.3 *(Joint differential privacy for dynamical systems) Let the privacy parameter $\varepsilon \geq 0$ be given and let $\mathcal{M}$ be a mechanism whose output is an N-tuple. Then $\mathcal{M}$ is ε-joint differentially private if and only if, for any $i \in [N]$, for all $u, \tilde{u} \in \tilde{\ell}_p^s$ satisfying $\mathrm{Adj}_B^i(u, \tilde{u}) = 1$, all times k, and*

all $A \in \mathcal{B}^{(k+1)(n-n_i)}$, $\mathcal{M}$ satisfies

$$\mathbb{P}\big((\mathcal{M}_{-i}(u))_{0:k} \in A\big) \leq e^{\varepsilon}\mathbb{P}\big((\mathcal{M}_{-i}(\tilde{u}))_{0:k} \in A\big),$$

where $\mathcal{B}^d$ is the Borel sigma-algebra on $\mathbb{R}^d$. ■

Joint differential privacy guarantees that when agent i's input changes by a small amount, the outputs corresponding to other agents do not change by much. A useful characteristic of both ordinary and joint differential privacy is their resilience to post-processing, which guarantees that post-hoc transformations of private data cannot weaken the privacy guarantees afforded to that data. This result is formalized below.

Lemma 6.4 *(Resilience to post-processing; [81, Proposition 2.1]) Let M be an ε-differentially private mechanism and let f be a function such that the composition $f \circ M$ is well-defined. Then $f \circ M$ is also ε-differentially private.* ■

6.2.3 The Laplace Mechanism

To enforce differential privacy for a particular choice of ε, noise must be added somewhere in the system, and the distribution of this noise must now be determined. One common mechanism in the literature draws noise from the Laplace distribution, used in both [79] and [82], and this mechanism is used to provide ε-joint differential privacy. To define this mechanism, the notion of the ℓ_p sensitivity of a system is now introduced.

Definition 6.2 *The ℓ_p sensitivity of a deterministic, causal system $\mathcal{G}$ is defined as*

$$\Delta_p\mathcal{G} = \sup_{u,\tilde{u}:\mathrm{Adj}_B(u,\tilde{u})=1} \|\mathcal{G}(u) - \mathcal{G}(\tilde{u})\|_{\ell_p}.$$

◊

The Laplace mechanism is stated in terms of the ℓ_1 sensitivity of a system. Below, the notation

$\text{Lap}(b)$ denotes a scalar Laplace distribution with mean zero and scale parameter b, i.e.,

$$\text{Lap}(b) := \frac{1}{2b} \exp\left(-\frac{|x|}{b}\right).$$

Lemma 6.5 *(Laplace mechanism; [82, Theorem 4]) Let $\varepsilon \geq 0$ be given. The Laplace mechanism defined by*

$$M(u) = \mathcal{G}(u) + w$$

with $w(k) \sim \text{Lap}(b)^r$ is ε-differentially private for $b \geq \Delta_1 \mathcal{G}/\varepsilon$, with r the dimension of the output space of the system. ■

Lemma 6.5 says that one can make a system private by adding noise drawn from a Laplace distribution to that system's output at each timestep. This idea is implemented for Problem 6.1 in the next section.

6.3 Optimizing Under Joint Differential Privacy

In this section, the privacy results in Section 6.2 are applied to Problem 6.1, and Algorithm 6.1 is made joint differentially private. Formally, g and g_{x_i} are treated as memoryless dynamical systems, and joint differential privacy is used to ensure that they keep the agents' state trajectories, which are the inputs to these systems, private.

6.3.1 Stochastic Optimization Algorithm

From Lemma 6.3, we see that enforcing joint differential privacy for the states in the network requires that the cloud make $q_i(k) := g_{x_i}\big(x(k)\big)^T \mu(k)$ private before it is sent to agent i. To make $g_{x_i}(x(k))$ private, noise can be added to it directly and this is done below. To make $\mu(k)$ private, we use the fact that computing $\mu(k)$ relies on $g(x(k-1))$, and add noise to make $g(x(k-1))$ private. Then computing $\mu(k)$ is joint differentially private by the post-processing property in Lemma 6.4.

Similarly, if the noisy forms of both $g_{x_i}(x(k))$ and $\mu(k)$ are joint differentially private, their product is as well, again by Lemma 6.4. Adding noise in this way, Algorithm 6.1 is modified to state the joint differentially private optimization algorithm below.

Algorithm 6.2

Step 0: *For all $i \in [N]$, initialize agent i with $x_i(0)$, X_i, f_i, $\{\alpha_k\}_{k\in\mathbb{N}}$, and $\{\gamma_k\}_{k\in\mathbb{N}}$. Initialize the cloud with $\bar{x}$, g, f_{lower}, $\{\alpha_k\}_{k\in\mathbb{N}}$, and $\{\gamma_k\}_{k\in\mathbb{N}}$. Let the cloud compute M before the system begins optimizing. Set $k = 0$.*

Step 1: *For all $i \in [N]$, the cloud computes*

$\tilde{q}_i(k) := \big(g_{x_i}\big(x(k)\big) + w_i(k)\big)^T \mu(k)$ *and sends it to agent i.*

Step 2: *Agent i computes*

$$x_i(k+1) = \Pi_{X_i}\Big[x_i(k) - \gamma_k\left(\nabla_i f_i\big(x_i(k)\big) + \tilde{q}_i(k) + \alpha_k x_i(k)\right)\Big],$$

and sends a state value $x_i(k+1)$ to the cloud.

Step 3: *The cloud computes*

$$\mu(k+1) = \Pi_M\left[\mu(k) + \gamma_k\left(L_\mu(k) + w_g(k) - \alpha_k\mu(k)\right)\right].$$

Step 4: *Set $k := k+1$ and return to Step 1.* $\triangle$

To solve Problem 6.1, Algorithm 6.2 must implement joint differential privacy using the noise terms w_g and w_i, while still converging to a minimum. The following theorem gives conditions on w_g and each w_i under which convergence to a minimum is guaranteed. Then Theorem 6.2 shows that joint differential privacy is achieved under these conditions, and Section 6.4 shows that each agent's incentive for misreport is indeed limited due to joint differential privacy.

Theorem 6.1 *Let $(\hat{x}, \hat{\mu})$ denote the least-norm saddle point of L. Algorithm 6.2 satisfies*

$$\lim_{k\to\infty} \mathbb{E}\big[\|x(k) - \hat{x}\|_2^2\big] = 0 \text{ and } \lim_{k\to\infty} \mathbb{E}\big[\|\mu(k) - \hat{\mu}\|_2^2\big] = 0$$

if i. $w_i(k)$ *and* $w_g(k)$ *have zero mean and bounded variance for all* k

ii. $\alpha_k = \bar{\alpha}k^{-c_1}$ *and* $\gamma_k = \bar{\gamma}k^{-c_2}$*, where* $0 < c_1 < c_2$*,* $c_1 + c_2 < 1$*,* $\bar{\alpha} \in (0,1)$*, and* $\bar{\gamma} \in (0,1)$

Proof: See [91, Theorem 6]. ■

It remains to be shown that Condition i of Theorem 6.1 can be satisfied when joint differential privacy is implemented, and this is done next.

6.3.2 Calibrating Noise for Joint Differential Privacy

Here the systems being made private are g and g_{x_i}, and the mechanisms acting for joint differential privacy add noise to $g(x(k-1))$ when computing $\mu(k)$ and add noise to $g_{x_i}(x(k))$ when computing $\tilde{q}_i(k)$. It was shown in Section 6.2 that the noise added in Algorithm 6.2 will enforce ε-joint differential privacy as long as it has large enough variance. To determine the variance of noise that must be added by the Laplace mechanism, bounds are derived on the ℓ_1 sensitivity of each g_{x_i} and g below.

Lemma 6.6 *For the relation* Adj_B*, the* ℓ_1 *sensitivities of* g_{x_i} *and* g *satisfy* $\Delta_1 g_{x_i} \leq L_{g,i}B$ *and* $\Delta_1 g \leq K_g B$*.*

Proof: See [107]. ■

Using Lemmas 6.5 and 6.6, we see that if $w_i(k) \sim \text{Lap}(b_i)$ with $b_i \geq \Delta_1 g_{x_i}/\varepsilon$ and $w_g(k) \sim \text{Lap}(b_g)$ with $b_g \geq \Delta_1 g/\varepsilon$ for all k, then all states are afforded (ordinary) ε-differential privacy in Algorithm 6.2. It turns out that this privacy and its resilience to post-processing imply that ε-joint differential privacy holds as well, which is stated formally in the following theorem.

Theorem 6.2 *Consider the mechanism* $\mathcal{M}$ *defined by*

$$\mathcal{M}\big(x(k)\big) = (\tilde{q}_1(k), \ldots, \tilde{q}_N(k)),$$

with $\tilde{q}_i(k) := \big(g_{x_i}\big(x(k)\big) + w_i(k)\big)^T \mu(k)$ *as defined in Algorithm 6.2 and* $\mu(k)$ *computed as in Algorithm 6.2. At each time* k*, if* $w_i(k) \sim \text{Lap}(b_i)$ *with* $b_i \geq \Delta_1 g_{x_i}/\varepsilon$ *and* $w_g(k) \sim \text{Lap}(b_g)$ *with* $b_g \geq \Delta_1 g/\varepsilon$*, then* $\mathcal{M}$ *is* ε*-joint differentially private.*

Proof: We examine $\mathcal{M}_{-i}$ for an arbitrary $i \in [N]$ whose output at time k is

$$\mathcal{M}_{-i}\big(x(k)\big) = (\tilde{q}_1(k), \ldots, \tilde{q}_{i-1}(k), \tilde{q}_{i+1}(k), \ldots, \tilde{q}_N(k)) \,.$$

Examining some $j \in [N]\backslash\{i\}$, the j^{th} output $\mathcal{M}_j\big(x(k)\big)$ is

$$\mathcal{M}_j\big(x(k)\big) = \tilde{q}_j(k) := \Big(g_{x_j}\big(x(k)\big) + w_j(k)\Big)^T \mu(k).$$

In light of the fact that $w_g(k-1) \sim \text{Lap}(b_g)$ with $b_g \geq \Delta_1 g/\varepsilon$, we see that $\mu(k)$ keeps $x(k-1)$ ε-differentially private. Similarly, $g_{x_j}\big(x(k)\big) + w_j(k)$ keeps $x(k)$ ε-differentially private because $w_j(k) \sim \text{Lap}(b_j)$ and $b_j \geq \Delta_1 g_{x_j}/\varepsilon$. By Lemma 6.4, the product $\tilde{q}_j(k) = \Big(g_{x_j}\big(x(k)\big) + w_j(k)\Big)^T \mu(k)$ keeps x ε-differentially private because it is the result of post-processing two differentially private quantities. By the same reasoning,

$$\mathcal{M}_{-i}\big(x(k)\big) = \big(\tilde{q}_1(k), \ldots, \tilde{q}_{i-1}(k), \tilde{q}_{i+1}(k), \ldots, \tilde{q}_N\big)$$

simply post-processes differentially private information. From Lemma 6.3 we conclude that $\mathcal{M}$ is ε-joint differentially private. ■

The next section describes the application of this mechanism to inducing approximately-truthful behavior in multi-agent optimization through the computation of β-approximate minima.

6.4 Computing β-approximate Minima

This section presents the main result of the chapter: joint differential privacy results in there being only minimal incentive for an agent to misreport its state to the cloud because it limits the reduction in cost an agent can attain through misreport. Toward showing this result, we first present the following lemma that bounds agent i's influence upon μ over time.

Lemma 6.7 *For two trajectories x and $\tilde{x}$ such that $\text{Adj}_B^i(x, \tilde{x}) = 1$ holds, suppose that agent i has misreported its state until some timestep k. Then, for the resulting dual vectors μ and $\tilde{\mu}$*

(corresponding to x and $\tilde{x}$, respectively), we find

$$\|\mu(k) - \tilde{\mu}(k)\| \leq \gamma_0 K_g B,$$

where γ_0 is the initial stepsize in Algorithm 6.2, K_g is the Lipschitz constant of g, and B is the adjacency parameter.

Proof: Without loss of generality, we consider the case in which agent i has begun misreporting at timestep 1. Because $x(0)$ and $\mu(0)$ are fixed, the effects of this misreport are not seen until timestep 2. In this case, we find

$$\begin{aligned}
\|\mu(2) - \tilde{\mu}(2)\| &= \|\Pi_M \left[\mu(1) + \gamma_1 \left(g(x(1)) - \alpha_1 \mu(1)\right)\right] - \Pi_M \left[\mu(1) + \gamma_1 \left(g(\tilde{x}(1)) - \alpha_1 \mu(1)\right)\right]\| \\
&= \|\mu(1) + \gamma_1 \left(g(x(1)) - \alpha_1 \mu(1)\right) - \mu(1) - \gamma_1 \left(g(\tilde{x}(1)) - \alpha_1 \mu(1)\right)\| \\
&= \gamma_1 K_g \|x(1) - \tilde{x}(1)\|,
\end{aligned}$$

where we have used the non-expansive property of $\Pi_M[\cdot]$ with respect to the Euclidean norm on $\mathbb{R}^m$. Applying a simple inductive argument then gives

$$\|\mu(k) - \tilde{\mu}(k)\| \leq K_g \sum_{j=1}^{k-1} \|x(j) - \tilde{x}(j)\| \cdot \gamma_j \prod_{p=j+1}^{k-1} (1 - \gamma_p \alpha_p).$$

Using that $\gamma_0 \in (0, 1)$ and $\alpha_0 \in (0, 1)$, along with the fact that both sequences are decreasing, we see that

$$\prod_{p=j+1}^{k-1} (1 - \gamma_p \alpha_p) < 1$$

for all j. Using this fact and that $\{\gamma_k\}_{k \in \mathbb{N}}$ is a decreasing sequence, we find

$$\|\mu(k) - \tilde{\mu}(k)\| \leq \gamma_0 K_g \sum_{j=1}^{k-1} \|x(j) - \tilde{x}(j)\|,$$

where using the adjacency of x and $\tilde{x}$ completes the proof. ■

Using Lemma 6.7, we next bound the change in cost agent i can attain through misreporting its state in the absence of any other agents.

Lemma 6.8 *Let the hypotheses of Lemma 6.7 hold. Then, at time k, we have*

$$\left|f_i\big(x_i(k)\big) - f_i\big(\tilde{x}_i(k)\big)\right| \leq \theta_i,$$

where

$$\theta_i = K_i\big[\big(\gamma_0 L_i + \alpha_0 + \mu_{max} L_g\big)B + k\gamma_0 K_g B\big],$$

with L_g the Lipschitz constant of ∇g, K_i the Lipschitz constant of f_i, L_i the Lipschitz constant of f_{x_i}, and $\mu_{max} = \sup_{\mu\in M} \|\mu\|_2$.

Proof: Using an inductive argument similar to that of Lemma 6.7, we find

$$\|x_i(k) - \tilde{x}_i(k)\| \leq \sum_{\ell=2}^{k-1} \Big[\big(\gamma_\ell L_i + \alpha_\ell\big)\|x_i(\ell) - \tilde{x}_i(\ell)\| + \|\nabla g\big(x(\ell)\big)\| \cdot \|\mu(\ell) - \tilde{\mu}(\ell)\| + \|\tilde{\mu}(\ell)\|\|\nabla g\big(x(\ell)\big) - \nabla g\big(\tilde{x}(\ell)\big)\|\Big].$$

Using the Lipschitz property of ∇g and the definition of μ_{max}, we then have

$$\|x_i(k) - \tilde{x}_i(k)\| \leq \sum_{\ell=2}^{k-1} \left[(\gamma_\ell L_i + \alpha_\ell)\|x_i(\ell) - \tilde{x}(\ell)\| + K_g\|\mu(\ell) - \tilde{\mu}(\ell)\| + \mu_{max} L_g\|x_i(\ell) - \tilde{x}_i(\ell)\|\right],$$

where applying Lemma 6.7 and using the adjacency of x and $\tilde{x}$ completes the proof. ■

In Lemma 6.8, the factor k counts the number of timesteps at which agent i has misreported its state. In the problems we are interested in, this will be a finite number because the adjacency parameter B is finite, giving agent i bounded influence upon its own cost in the absence of other agents. In addition, the influence of agent i can be tuned by careful selection of γ_0 and α_0. Furthermore, by scaling the inequality constraints g, one can leave intact the conditions under which these constraints are met, while changing its Lipschitz constant K_g to affect the value of θ_i. The

ability to change θ_i is important in our main results, which will be presented below. We note that Lemma 6.8 concerns agent i's ability to affect its own cost *in the absence of other agents*; the main result concerns agent i's ability to affect its cost in Algorithm 6.2, which will be presented shortly.

Toward showing that result, a uniform upper bound on f_i over X_i is first presented.

Lemma 6.9 *Let $\bar{x}$ denote a Slater point for g. Then for all $i \in [N]$, $f_i(x_i) \leq \lambda_i$ for all $x_i \in X_i$, where $\lambda_i := f_i(\bar{x}_i) + K_i D_i$.*

Proof: Using the Mean-Value Theorem and the Cauchy-Schwarz inequality, we have

$$\begin{aligned} f_i(x_i) &= f_i(\bar{x}_i) + \nabla f_i(z_i)^T (x_i - \bar{x}_i) \\ &\leq f_i(\bar{x}_i) + \|\nabla_i f_i(z_i)\| \cdot \|x_i - \bar{x}_i\|, \end{aligned}$$

for some $z_i \in X_i$. The result follows by bounding $\|\nabla_i f_i(z_i)\|$ by K_i and bounding $\|x_i - \bar{x}_i\|$ by D_i. ■

We now present our main results for the chapter. Below, each expected value is over the randomness introduced by the mechanism $\mathcal{M}$. For clarity, this theorem tracks the state agent i has reported to the cloud and the state trajectory of every *other* agent. The notation $\mathbb{E}[f_i\big(x_i(k)\big)|y_i, v_{-i}]$ is used to denote agent i's expected cost at time k when agent i has reported the trajectory y_i to the cloud and every other agent has followed the trajectory v. The symbol x_i is always used as the argument to f_i because f_i always depends on the true state of agent i, not the state it reports.

Theorem 6.3 *Suppose Assumptions 6.1 and 6.2 hold. Let the agents and cloud execute Algorithm 6.2 with the cloud implementing the mechanism $\mathcal{M}$. Then for $\varepsilon \in (0, 1)$, all agents sharing their true states in Algorithm 6.2 results in a β-approximate minimum. In particular, at all times k and for any state trajectories $x, \tilde{x} \in \tilde{\ell}_p^s$ satisfying $\text{Adj}_B^i(x, \tilde{x}) = 1$, we have*

$$\mathbb{E}[f_i(x_i(k))|x_i, x_{-i}] \leq \mathbb{E}[f_i(x_i(k))|\tilde{x}_i, \tilde{x}_{-i}] + \beta,$$

where $\beta = 2\varepsilon\lambda_i + \theta_i$, and where agent i is misreporting its state in $\tilde{x}$.

Proof: From Lemma 6.8 we have

$$\mathbb{E}[f_i(x_i(k))|x_i, x_{-i}] \leq \mathbb{E}[f_i(x_i(k))|\tilde{x}_i, x_{-i}] + \theta_i. \tag{6.5}$$

Using Theorem 6.2 and the definition of ε-joint differential privacy we find

$$\mathbb{E}[f_i(x_i(k))|\tilde{x}_i, x_{-i}] \leq e^{\varepsilon}\mathbb{E}[f_i(x_i(k))|\tilde{x}_i, \tilde{x}_{-i}],$$

which we substitute into Equation (6.5) to find

$$\mathbb{E}[f_i(x_i(k))|x_i, x_{-i}] \leq e^{\varepsilon}\mathbb{E}[f_i(x_i(k))|\tilde{x}_i, \tilde{x}_{-i}] + \theta_i.$$

Using $e^{\varepsilon} \leq 1 + 2\varepsilon$ for $\varepsilon \in (0, 1)$ gives

$$\mathbb{E}[f_i(x_i(k))|x_i, x_{-i}] \leq \mathbb{E}[f_i(x_i(k))|\tilde{x}_i, \tilde{x}_{-i}] + 2\varepsilon\mathbb{E}[f_i(x_i(k))|\tilde{x}_i, \tilde{x}_{-i}] + \theta,$$

where we apply Lemma 6.9 to get

$$\mathbb{E}[f_i(x_i(k))|x_i, x_{-i}] \leq \mathbb{E}[f_i(x_i(k))|\tilde{x}_i, \tilde{x}_{-i}] + 2\varepsilon\lambda_i + \theta_i.$$

■

While the θ_i term in β is a feature of the problem itself, the $\varepsilon\lambda_i$ term results directly from the untruthfulness of agent i, and it is precisely this term which can be influenced using the privacy parameter ε, allowing a network operator to directly counteract the influence of false information. Of course, shrinking ε requires that more noise be added which, in general, degrades performance in the system. One must therefore balance the two objectives of incentivizing truthful information sharing and system performance based upon the needs in a particular application.

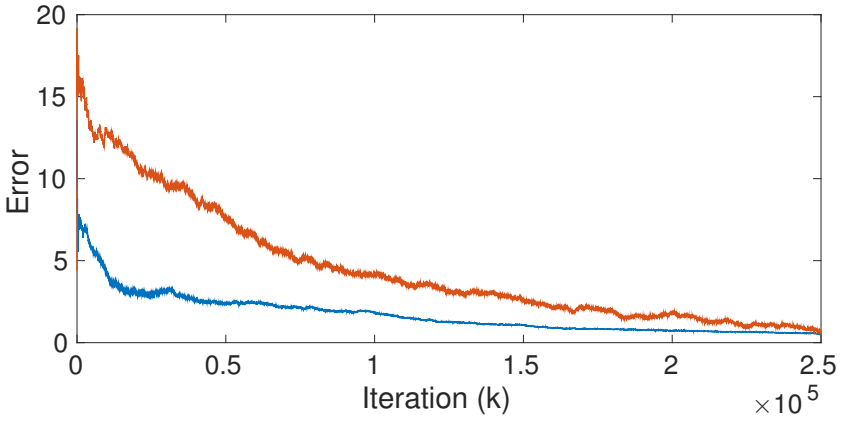

Figure 6.2: A plot of the distance to the saddle point in Problem 6.0 in the primal space (lower curve) and dual space (upper curve) when all agents are truthful.

6.5 Simulation Results

A simulation was run consisting of $N = 8$ agents, each with $x_i \in \mathbb{R}^2$, and $m = 4$ constraints. The constraint $X_i = [-10, 10]^2$ for all $i \in [N]$, and $f_i(x_i) = \frac{1}{2}\|x_i - t_i\|_2^2$, for all $i \in [N]$; the value of each t_i can be found in Table 6.1. The constraints used were

$$g(x) = \begin{pmatrix} \|x_1 - x_2\|_2^2 + \|x_1 - x_3\|_2^2 - 5 \\ \|x_4 - x_5\|_2^2 + \|x_4 - x_6\|_2^2 - 3 \\ \|x_7 - x_8\|_2^2 + \|x_7 - x_6\|_2^2 - 3 \\ \|x_5 - x_3\|_2^2 + \|x_5 - x_7\|_2^2 - 5 \end{pmatrix}.$$

The value $B = 3$ was used for adjacency and the privacy parameter was chosen to be $\varepsilon = \log 3$. The distributions of each w_i are shown in Table 6.1; in addition to those values, $w_g \sim \text{Lap}(327.69)$. The stepsize and regularization parameters were chosen to be $\gamma_k = 0.01k^{-3/5}$ and $\alpha_k = 0.5k^{-1/3}$. Two adjacent problems were run for $250,000$ timesteps each, with agent 6 being untruthful in one of them. In that run, agent 6 reported its unconstrained minimizer t_6 to the cloud at each timestep instead of its actual state.

Figure 6.2 plots the distance to the saddle point in the primal and dual spaces when all agents

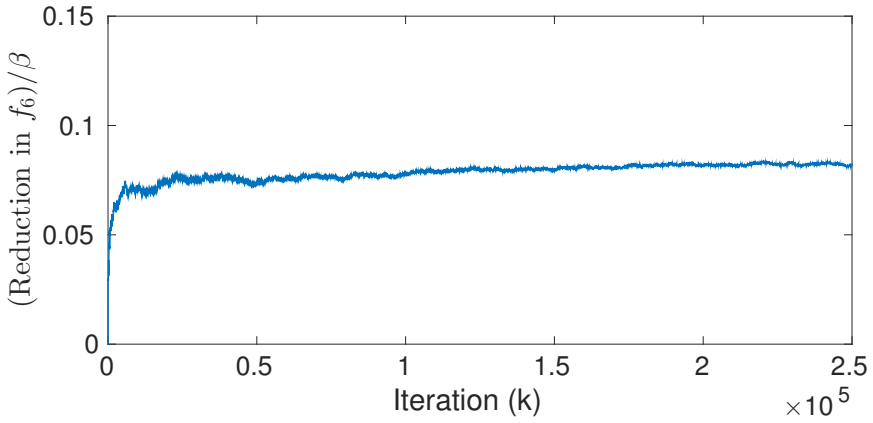

Figure 6.3: A plot of the decrease in cost agent 6 attains through misreporting its state when using Algorithm 6.2. The ordinate is normalized by β.

Table 6.1: Values of t_i and distributions of w_i, $i \in [8]$, for implementing joint differential privacy in Algorithm 6.2.

i	t_i	Distribution of w_i
1	$[6\ \ -4]^T$	Lap(10.92)
2	$[2\ \ 2]^T$	Lap(5.46)
3	$[-7\ \ 7]^T$	Lap(5.46)
4	$[8\ \ -9]^T$	Lap(10.92)
5	$[3\ \ -7]^T$	Lap(16.38)
6	$[10\ \ 10]^T$	Lap(10.92)
7	$[-10\ \ -10]^T$	Lap(16.38)
8	$[6\ \ -6]^T$	Lap(5.46)

are truthful. The final error values in that case were

$$\|x(250,000) - \hat{x}\|_2 = 0.5367 \text{ and } \|\mu(250,000) - \hat{\mu}\|_2 = 0.6870,$$

indicating close convergence in each space despite the large amount of noise in the system.

Figure 6.3 plots the decrease in cost agent 6 sees in misreporting its state. This decrease is not only bounded by β, but is bounded above by 0.1β for all time, indicating that β may be loose in some cases; this is not surprising given that β uses Lipschitz constants and set diameters, which are "worst-case" in the sense that they give maximum values over all possible states. Nonetheless, the algorithm can be seen both to converge and compute a β-approximate minimum, indicating that Problem 6.1 has been solved.

6.6 Conclusion

It was shown in this chapter that joint differential privacy can be used in multi-agent optimization to incentivize truthful information sharing. Applications of the work presented here include any multi-agent setting in which the iterates of an optimization algorithm correspond to some physical quantity of interest. It is a feature of this work that, in conjunction with the work in Chapter 5 one can implement both differential privacy for state trajectories and joint differential privacy, simultaneously protecting user data and disincentivizing untruthful information reporting in the network.

Future applications of this work include the smart power grid, where agents' states correspond to their power usage levels. A user may be tempted to untruthfully report their power usage to the cloud over time, e.g., by reporting lower usage during a power-intensive task such as charging an electric car. Such a scenario is accounted for by this work, and its ability to reach solutions in the presence of noise makes it well-suited to such applications.

CHAPTER 7

PRIVATE OBJECTIVE FUNCTIONS IN MULTI-AGENT OPTIMIZATION

Distributed and multi-agent optimization problems arise naturally in a number of disparate fields, including power systems [65], [108], communications [9], [109], [110], signal processing [111], [112], sensor networks [2], [3], and machine learning [26], [113]. In these applications, it is common to have agents in a network directly share information with each other in the process of optimizing in order to guarantee that information flows across a network [37].

However, in some networked systems, users may be required to share potentially sensitive information, such as their locations over time. As a result, there are substantial privacy concerns associated with information sharing in some multi-agent systems, though some amount of information sharing is still required to successfully coordinate a network. To satisfy both a global network objective and the privacy requirements of individual users, differential privacy has been applied in a number of ways throughout the literature [78], [80]. At a high level, a user's information is differentially private if a change in that user's data does not produce significant changes in the quantities visible to other participants in the network or anyone observing it.

The wide use of multi-agent optimization and the flexibility of differential privacy have naturally led to interest in crafting differentially private optimization algorithms for use in network coordination problems [85]–[87], [114]. Broadly, these algorithms seek to keep individual users' data private while still producing results which are "close enough" to those obtained in a non-private problem formulation. Often, there is a trade-off between privacy and convergence in these implementations, and one often incurs worse network performance in exchange for increased privacy. This fundamental trade-off between privacy and performance is a common phenomenon in the broader context of privacy apart from optimization [115], [116]. While there are a number of

existing works focused on providing privacy to agents' states, in this chapter we study the problem of keeping each agent's objective function differentially private. While an agent's state can reveal an agent's location and perhaps what it is doing, an agent's objective function can reveal what it values and, as a result, what it is thinking or intending to do. For example, knowing that someone is trying to minimize the cost or time required to travel to some destination gives a strong indication of their future location. A key technical hurdle in this setting is that adding independent noise at each point in time can only keep objective functions differentially private across a finite time horizon [85]; across an infinite horizon, adding independent noise at each point in time will cause these noisy terms to cancel out and reveal users' objective functions [117]. This is highly undesirable as convergence of an optimization algorithm is often asymptotic, requiring, in general, an infinite time horizon. To circumvent this issue, a technique for directly perturbing objective functions has been developed which keeps them private while maintaining certain smoothness properties [117]. This method is suitable for a distributed implementation, and indeed was designed with such problems in mind, though it applies only to strongly convex objective functions. Work presented in [85] is likewise well-suited to distributed problems and focuses specifically on privacy for additive terms in affine objective functions.

In this chapter, we are interested in keeping objective functions private in multi-agent linear programs, and we present results designed for such problems. We propose using a central aggregator for protecting agents' objective functions across infinite time horizons, and this aggregator adds completely correlated noise over time. As in earlier chapters, the central aggregator will be thought of as a cloud computer, and the use of cloud computing is inspired by real-world systems in which the flexibility of cloud computing has been leveraged across a wide range of applications. In the cloud-based architecture, each agent possesses an objective function to minimize, while the agents are collectively subject to global constraints that encode ensemble-level requirements upon the agents. To jointly coordinate their activities while keeping their objective functions private, the agents do not directly share state information with each other. Instead, they route messages only through a trusted cloud computer which also enforces differential privacy for the agents, and it will

be shown that this configuration allows for private networked coordination, with only small errors introduced by incorporating privacy into the problem.

The rest of the chapter is organized as follows. In Section 7.1, a general form of the desired optimization algorithm is presented, without privacy, in order to define the update law used and the role of the cloud. Then, in Section 7.2, we incorporate privacy into the cloud-based distributed optimization algorithm by adding noise, and we give the definition of differential privacy used for objective functions in this setting. In Section 7.3, we design a noise-adding mechanism for the cloud-based distributed algorithm that provides privacy guarantees to users' objective functions. In Section 7.4, the trade-off between privacy of objective functions and the performance of the noisy optimization algorithm is studied for the noise-adding mechanism proposed. Section 7.5 then presents simulation results to demonstrate the bounds on performance we derive. Finally, Section 7.6 concludes the chapter.

7.1 Problem Formulation

This section describes the multi-agent optimization problems to be solved and details the assumptions imposed upon them. In this chapter, we use the notation $[n] = \{1, \dots, n\}$ for a natural number n. A finite sequence indexed from 1 to T is denoted $v^{(T)} = \{v(1), \dots, v(T)\}$. We also use the notation $\mathbb{R}_+$ to denote the set of non-negative real numbers, and we use $\mathbb{R}_+^{\mathbb{N}}$ to denote the set of infinite sequences with non-negative real entries. We say that a sequence $\{v(t)\}_{t=1}^{\infty} \in \mathbb{R}_+^{\mathbb{N}}$ is *dominated* by another infinite sequence $\{u(t)\}_{t=1}^{\infty} \in \mathbb{R}_+^{\mathbb{N}}$ as $t \to \infty$ if $\lim_{t\to\infty} v/u \leq 1$, and we express this condition with the notation $v \preceq u$. A finite sequence of 0 is also denoted by 0 when there is no ambiguity. In addition, the set of second-order continuously differentiable convex functions is denoted by C^2.

7.1.1 Distributed Optimization

Consider a discrete-time multi-agent system of N agents indexed over $I = \{1, 2, \dots, N\}$. The state of the i^{th} agent is denoted x_i, and it takes values in a non-empty, compact, convex set

$X_i \subseteq \mathbb{R}^{n_i}$, where $n_i \in \mathbb{N}$. The ensemble state of the system is given by

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix} \in X \subseteq \mathbb{R}^n,$$

where $n = \sum_{i=1}^{N} n_i$ and $X = \prod_{i=1}^{N} X_i$.

Each agent seeks to minimize a local C^2 objective function

$$f_i : \mathbb{R}^{n_i} \to \mathbb{R}, \quad i \in [N],$$

which depends only upon its own state, while subject to m global C^2 constraints of the form $g_j(x) \leq 0$, where

$$g_j : \mathbb{R}^n \to \mathbb{R}, \quad j \in [m].$$

As with some other related works, we make the following assumption on the constraints.

Assumption 7.1 *The constraints satisfy Slater's condition: there exists an ensemble state $\bar{x} \in X$ such that every constraint is strictly satisfied, namely $g_j(\bar{x}) < 0$ for all $j \in [m]$.* $\lozenge$

As in some other works on differentially private optimization [118], [119], we require that the constraints be Lipschitz continuous, which we formally state below.

Assumption 7.2 *For each $j \in [m]$ and $i \in [N]$, there exist constants $l_{i,j} \in \mathbb{R}_+$ such that*

$$\left\| \frac{\partial g_j}{\partial x_i} \right\| \leq l_{i,j}.$$

We call $l_{i,j}$ the i^{th} Lipschitz constant for g_j, and $l_j = \max_{i \in [N]} l_{i,j}$ the overall Lipschitz constant for g_j. $\lozenge$

Due to the fact that the objective functions are local while the constraints are global, we employ a cloud-based communication architecture to coordinate the agents while they optimize [120].

Specifically, the agents do not communicate directly with each other; instead, to enforce privacy of the agents' objective functions, they send messages only through the cloud. At each timestep, the agents receive information from cloud, update their states, and send these updated states back to the cloud. The cloud then gathers the new states and computes new information required by the agents in their next state updates. It will be shown in Section 7.2 how these computations in the cloud are made private.

For the whole system, optimizing every $f_i(x_i)$ simultaneously is equivalent to optimizing the global objective function

$$f(x) = \sum_{i=1}^{N} f_i(x_i).$$

In terms of f, g, and X, one can define an ensemble-level optimization problem, shown below.

Problem 7.1

$$\begin{aligned} &\text{minimize } f(x) \\ &\text{subject to } g(x) \leq 0 \\ &x \in X. \end{aligned}$$

♦

We use a seminal result in non-linear programming to transpose this problem to an equivalent form more amenable to being solved on the cloud-based system we use. In particular, using [29], we form the Lagrangian associated with Problem 7.1 as

$$L(x, \mu) = f(x) + \mu^{\mathrm{T}} g(x),$$

where μ is a vector of Kuhn-Tucker (KT) multipliers that takes values in $M = \mathbb{R}^m_+$. With this definition, we minimize f subject to $g(x) \leq 0$ by finding a saddle point of L [29]. In particular, we use the gradient-based algorithm of Bakushinskii and Polyak presented in [34], which takes the

form

$$x \leftarrow \Pi_X \left[x - \gamma_t \left(\frac{\partial f}{\partial x}(x) + \frac{\partial g}{\partial x}(x)^T \mu + \alpha_t x \right) \right] \tag{7.1}$$

$$\mu \leftarrow \Pi_M \left[\mu + \gamma_t \left(g(x) - \alpha_t \mu \right) \right]. \tag{7.2}$$

Here, the time-dependent parameters α_t and γ_t asymptotically vanish, and their precise form will be given below in Theorem 7.1. We choose this algorithm as the basis for the current work due to the noise-rejecting properties it provides and the ability to naturally distribute it over a network, and these properties will be exploited when noise is added for privacy below.

In this problem we will consider separable constraints. A careful inspection of Equations (7.1) and (7.2) shows that this algorithm is therefore compatible with the cloud-based architecture because separability of g implies than an agent only needs to receive μ from the cloud. The cloud can easily compute and share μ with each agent precisely because it receives global information about the network at each timestep. While Equation (7.2) is computed by the cloud in a centralized way, Equation (7.1) can be computed locally by each agent given μ. With this in mind, we now state the cloud-based optimization algorithm as Algorithm 7.1.

Algorithm 7.1 Non-Differentially Private Cloud-Based Optimization

Given $x_0 \in X$, $\mu_0 \in M$, parameters α_t, γ_t, times of iteration T.
$x_i \leftarrow x_i(0)$, $\mu \leftarrow \mu_0$
for $t = 1$ to T **do**
    $\mu' \leftarrow \Pi_M \left[\mu + \gamma_t \left(g(x) - \alpha_t \mu \right) \right]$
    for $i = 1$ to N **do**
        $x_i \leftarrow \Pi_{X_i} \left[x_i - \gamma_t \left(\frac{\partial f_i}{\partial x_i}(x_i) + \mu \frac{\partial g}{\partial x_i}(x_i) + \alpha_t x_i \right) \right]$
    end for
    $\mu \leftarrow \mu'$
    $t \leftarrow t + 1$
end for

The convergence of Algorithm 7.1 is guaranteed by the following theorem, derived from Theorem 6 in [91].

Theorem 7.1 *Algorithm 7.1 converges to an optimum as the times of iteration $T \to \infty$, if the time-dependent parameters α_t and γ_t satisfy*

$$\gamma_t = \gamma_1 t^{-c_1}, \quad \alpha_t = \alpha_1 t^{-c_2}, \tag{7.3}$$

where $\alpha_1, \gamma_1 > 0$, $\alpha_1\gamma_1 \in (0, 1)$, $c_1 > c_2 > 0$, and $c_1 + c_2 < 1$. ■

Convergence rates for this algorithm can range from linear to geometric [36] and depend upon the specific problem under consideration, as well as the choices of α_t and γ_t. In the next section, we explore how to incorporate privacy into the execution of this algorithm.

7.2 Differential Privacy for Objective Functions

This section provides the necessary background on differential privacy for the cloud-based implementation used in this chapter. Even when the agents communicate through the cloud, privacy is a major concern because an individual agent's information could be computed from the cloud's transmissions to other agents. A commonly used means for keeping data private is the framework of differential privacy. Unlike a number of other efforts to keep an agent's state x_i private, here we keep each agent's objective function f_i private using differential privacy.

In the cloud-based system, the information from the central server $\mu(t)$ is assumed to be publicly known information. Agent i's objective function f_i, which we want to keep private, is chosen from a parameterized family of functions

$$\mathcal{F}_i = \{f_i(\cdot; a_i) \mid a_i \in \mathcal{A}_i\}, \quad i \in [N],$$

where $\mathcal{A}_i$ is a set of parameters equipped with a metric

$$d_i : \mathcal{A}_i \times \mathcal{A}_i \to \mathbb{R}_+, \quad i \in [N].$$

Concerning the agents' objectives, we make two assumptions, stated below.

Assumption 7.3 *The parameterized family of objective functions $\mathcal{F}_i$ contains only linear objective functions, namely*

$$\mathcal{F}_i = \left\{ f_i(x) = a_i^{\mathrm{T}} x_i \mid a_i \in \mathbb{R}^{n_i} \right\}, \quad i \in [N].$$

The distance on the parameters inherits from the Euclidean norm on $\mathbb{R}^{n_i}$, namely

$$d_i(x, y) = \|x - y\|.$$

$\Diamond$

Next, we state an assumption on the constraints g.

Assumption 7.4 *There is only one linear constraint of the form*

$$g(x) = b^{\mathrm{T}} x = \sum_{i=1}^{N} b_i^{\mathrm{T}} x_i,$$

where $b = [b_1^{\mathrm{T}}, \dots b_N^{\mathrm{T}}]^{\mathrm{T}}$. The i^{th} Lipschitz constant for g is $l_i = \|b_i\|$, and the overall Lipschitz constant is $l = \max_{i \in [N]} l_i$. $\Diamond$

These two assumptions are enforced only to simplify the forthcoming presentation of results; the theoretical results presented in the following sections are still valid when the objective functions f_i and the constraint g are C^2 and Lipschitz continuous.

7.2.1 Noise-Adding Mechanism

To keep agents' objective functions differentially private, we consider a mechanism which directly adds noise to the public multiplier μ, as shown in Algorithm 7.2. At each iteration t, the cloud

adds zero mean noise to the public multiplier by computing

$$\mu \leftarrow \mu + v(t), \quad t \in [T]$$

before sending it to the agents. The noise $v(t)$ added over time can be correlated, and we will make use of this fact below.

Algorithm 7.2 Differentially Private Cloud-Based Distributed Optimization

Initialize $x_0 \in X$, $\mu_0 \in M$, parameters α_t, γ_t, number of iterations T.
$x_i \leftarrow x_i(0)$, $\mu \leftarrow \mu_0$
for $t = 1$ to T **do**
    $\mu' \leftarrow \Pi_M \left[\mu + \gamma_t \left(b^{\mathrm{T}} x - \alpha_t \mu\right)\right]$
    for $i = 1$ to N **do**
        $x_i \leftarrow \Pi_{X_i} \left[x_i - \gamma_t \left(a_i + \mu b_i + \alpha_t x_i\right)\right]$
    end for
    $\mu \leftarrow \mu' + v(t)$
    $t \leftarrow t + 1$
end for

Compared to some other mechanisms in the literature that add noise to the states or the objective functions of every agent, this mechanism is much easier to implement in practice because only the cloud needs to add noise to its computations, i.e., the agents' update laws are entirely unaffected by the inclusion of differential privacy in this problem.

7.2.2 Differential Privacy with Metric

Two key components of any differential privacy implementation are the sensitive data, i.e., the user data that we wish to keep private, and the observables of a system, which are the quantities that can be seen by another member of the network or an eavesdropper. The sensitive data in this setting are drawn from the set of objective functions $D = \{f_i(x_i) = a_i^{\mathrm{T}} x_i \mid i \in [N]\} \in \prod_{i=1}^{N} \mathcal{F}_i$. Two

sets of sensitive data are *adjacent* if they differ in at most one entry. In addition, we now define a distance between adjacent sets of sensitive data, which derives naturally from the Euclidean metric on the set of admissible objective functions.

Definition 7.1 *Two sets of sensitive data* $D = \{f_i(x_i) = a_i^{\mathrm{T}} x_i \mid i \in [N]\}$ *and* $D' = \{f_i'(x_i) = a_i'^{\mathrm{T}} x_i \mid i \in [N]\}$ *are adjacent if there exists an index* $i \in [N]$ *such that* $a_j = a_j'$ *for each* $j \in [N] \backslash \{i\}$. *In addition, the distance between the two* adjacent *datasets is defined by*

$$d(D, D') = \|a_i - a_i'\|,$$

where $\|\cdot\|$ *is the Euclidean norm on* $\mathbb{R}^{n_i}$. $\lozenge$

The observable in Algorithm 7.2 is the sequence of public multipliers $\mu^{(T)} = \{\mu(1), \ldots, \mu(T)\}$, where $\mu(t) \in M$ for all $t \in [T]$. It is determined by three factors:

- the choice of initial condition $x_0 \in X$, $\mu_0 \in M$,
- the noise-adding mechanism, namely the probability distribution of $v^{(T)} = \{v(1), \ldots, v(T)\}$ in Algorithm 7.2,
- the sensitive data D.

Remark 7.1 *When fixing the initial condition* $x_0 \in X$ *and* $\mu_0 \in M$, *the sensitive data* D, *and the noise-adding mechanism* $v^{(T)}$, $\mu^{(T)}$ *is a sequence in* $\mathbb{R}$, *which we denote by* $(\mu_{D,v^{(T)}}^{x_0,\mu_0})^{(T)}$; *when only fixing the initial condition* $x_0 \in X$ *and* $\mu_0 \in M$ *and the private data* D, $\mu^{(T)}$ *is a random process in* $\mathbb{R}$, *which we denote by* $(\mu_D^{x_0,\mu_0})^{(T)}$. $\lozenge$

In this work, we use the metric version of differential privacy from [121], defined below, which is stronger than the commonly used non-metric version of differential privacy.

Definition 7.2 *The cloud-based system is* (ε, δ)*-differentially private if, for any choice of initial condition* $x_0 \in X$ *and* $\mu_0 \in M$, *for any two sets of adjacent private data* D, D', *and for any*

possible set of observations $\mathcal{O}$, the inequality

$$\mathbb{P}\left[(\mu_D^{x_0,\mu_0})^{(T)} \in \mathcal{O}\right] \leq e^{\varepsilon d(D,D')}\mathbb{P}\left[(\mu_{D'}^{x_0,\mu_0})^{(T)} \in \mathcal{O}\right] + \delta$$

holds, where $(\mu_D^{x_0,\mu_0})^{(T)}$ and $(\mu_{D'}^{x_0,\mu_0})^{(T)}$ are random processes as explained in Remark 7.1. When $\delta = 0$, we call the distributed system ε-differentially private. $\Diamond$

7.2.3 Influence of Noise on Performance

Generally, adding noise $v^{(T)}$ in Algorithm 7.2 has an adverse effect upon convergence to optima in the cloud-based distributed optimization problem. We measure this *loss of performance* by the difference between the result obtained by Algorithm 7.2, which adds noise for privacy, and that obtained by Algorithm 7.1, which does not add noise.

Definition 7.3 *The loss of performance for sensitive data D in Algorithm 7.2 due to the noise added for privacy is defined by*

$$\Lambda_D = \max_{\substack{x_0 \in X \\ \mu_0 \in M}} \mathrm{Var}_{v^{(T)}}\left[\mu_{D,v^{(T)}}^{x_0,\mu_0}(T) - \mu_{D,0}^{x_0,\mu_0}(T)\right],$$

where $\mu_{D,v^{(T)}}^{x_0,\mu_0}(t)$ and $\mu_{D',0}^{x_0,\mu_0}(t)$ are the public multipliers at time T generated by Algorithm 7.2 with initial condition x_0, μ_0, set of objective functions D, and noises $v^{(T)}$ and 0, respectively. $\Diamond$

In the next section, we discuss our implementation of differential privacy, and, following that, we bound its resulting loss of performance.

7.3 Differentially Private Noise-Adding Mechanism

In this section, we detail the privacy mechanism used by the cloud to protect agents' objective functions. The difficulty of keeping objective functions differentially private in Algorithm 7.2 comes from the fact that they are used in computations at every iteration. As observations accumulate, the blurring effect of noise can therefore be weakened, possibly leading to losses of privacy. In

addition, the influence of perturbing an objective function even temporarily extends to other agents and remains over time, meaning that noise added to the system can have a substantial, long-lasting effect upon convergence. Furthermore, the influence of noise added to the multipliers to keep them differentially private also remains over time.

To account for these challenges while still successfully coordinating networks of agents, we analyze how the non-differentially private cloud-based optimization algorithm responds to perturbations of the agents' objective functions, and we then design noise to mask this perturbation. This approach is an extension to the commonly used *sensitivity analysis*. For two adjacent sets of objective functions, we view the difference between a_i and a_i' as a perturbation and analyze how this perturbation propagates through the system in order to gain insight into how two adjacent problems behave in Algorithm 7.2. With this insight, we design a privacy mechanism that masks the differences between the two problems as Algorithm 7.2 is executed.

7.3.1 Temporary Perturbations of Objective Functions

For a fixed $i \in [N]$, we study the case of perturbing the parameter a_i of the objective function $f_i(x_i) = a_i^{\mathrm{T}} x_i$ at time s, with no noise being added. In Algorithm 7.2, this is equivalent to applying the same perturbation directly to the state x_i at the same time, but scaled by $\gamma_t/(1-\alpha_t\gamma_t)$. Though this perturbation is only applied at time s, agent i computes a state update using the perturbed value of f_i and then shares this updated state with the cloud. This value in turn affects the computation of subsequent μ values, and these values themselves affect subsequent state updates made by other agents. Thus the influence of this perturbation persists over time.

Formally, setting $v^{(t)} = 0$, the update law of Algorithm 7.2 can be written as

$$\begin{bmatrix} x \\ \mu \end{bmatrix} \leftarrow \Pi_{X\times M}\left[\begin{bmatrix} (1-\alpha_t\gamma_t)I_n & -\gamma_t b \\ \gamma_t b^{\mathrm{T}} & 1-\alpha_t\gamma_t \end{bmatrix}\begin{bmatrix} x \\ \mu \end{bmatrix} - \begin{bmatrix} \gamma_t a \\ 0 \end{bmatrix}\right],$$

where $a = [a_1^{\mathrm{T}}, \ldots, a_N^{\mathrm{T}}]^{\mathrm{T}}$ and I_n is the $n \times n$ identity matrix. Therefore, by the non-expansive property of the projection $\Pi_{X\times M}$ with respect to the Euclidean norm on $\mathbb{R}^{n+m}$, when there is a

perturbation τ on the state x and a perturbation δ on the multiplier μ at time t, the perturbation propagated to the next iteration is bounded by

$$\left\| \begin{bmatrix} (1-\alpha_t\gamma_t)I_N & -\gamma_t b \\ \gamma_t b^{\mathrm{T}} & 1-\alpha_t\gamma_t \end{bmatrix} \begin{bmatrix} \tau \\ \delta \end{bmatrix} \right\|.$$

In this case, we start with the perturbation

$$\begin{aligned} \delta &= 0, \\ \tau &= [0, \ldots, 0, 1, 0, \ldots, 0]^{\mathrm{T}}, \end{aligned}$$

where 1 is the i^{th} element of τ (because we perturb a_i), and we want to design a mechanism that adds noise only to μ to dilute this perturbation's influence over time.

Indeed, to dilute the persistent influence of perturbing an objective function temporarily at time s, we add noise to the public multiplier μ over time. As shown in Figure 7.1, the perturbation in $x(s)$ propagates to both $x(s+1)$ and $\mu(s+1)$. This influence on $\mu(s+1)$ can be masked by directly adding Laplace noise to it, while the influence on $x(s+1)$ will be masked by adding noise to $\mu(s+2), \mu(s+3), \ldots$ in subsequent iterations as the effects of perturbing $x(s)$ propagate forward in time. Since all the influences are generated by a single perturbation, we choose to add completely correlated noise over time as there is only a single quantity to mask.

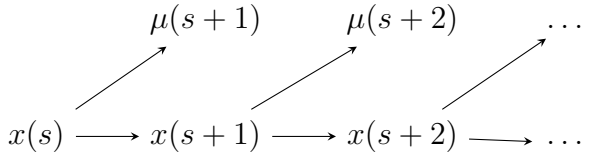

Figure 7.1: To cover the persistent influence of perturbing an objective function temporarily at time s, we add noise at each time $t \geq s+1$ that only covers the proportion of the perturbation that propagates through the lines shown here, thereby keeping each agent's objective function differentially private.

This idea yields the following differentially private mechanism to cover a temporary perturbation on an arbitrary objective function $f_i(x_i)$ at time s.

Mechanism 7.1 *In Algorithm 7.2, add completely correlated Laplacian noise*

$$v_s(t) = \begin{cases} 0, & \text{if } 1 \le t \le s \\ \gamma_s \gamma_{s+1} l w, & \text{if } t = s+1 \\ \gamma_s \gamma_t \left[\prod_{k=s+1}^{t-1}(1-\alpha_k\gamma_k)\right] l w, & \text{if } t \ge s+2 \end{cases},$$

where α_t, γ_t are given in Equation (7.3)*, l is the overall Lipschitz constant for the constraint g, and $w \sim \text{Lap}(1/\varepsilon)$ is a Laplacian noise.* △

The following lemma gives the asymptotic behavior of the noise $v_s(t)$ in Mechanism 7.1.

Lemma 7.1 *The noise $v_s(t)$ in Mechanism 7.1 obeys*

$$v_s(t) \le \gamma_1^2 (st)^{-c_1} \exp\left(-\alpha_1\gamma_1 \frac{t^{1-c_1-c_2} - s^{1-c_1-c_2}}{1-c_1-c_2}\right) l w,$$

when $t \to \infty$.

Proof:

For $t > s$ we have

$$\begin{aligned} \prod_{k=s}^{t-1}(1-\alpha_k\gamma_k) &= \exp\left(\sum_{k=s}^{t-1}\log(1-\alpha_k\gamma_k)\right) \\ &\le \exp\left(-\sum_{k=s}^{t}\alpha_k\gamma_k\right) \\ &= \exp\left(-\alpha_1\gamma_1\sum_{k=s}^{t}k^{-(c_1+c_2)}\right) \\ &\le \exp\left(-\alpha_1\gamma_1\frac{t^{1-c_1-c_2}-s^{1-c_1-c_2}}{1-c_1-c_2}\right), \end{aligned} \tag{7.4}$$

where the first inequality follows from the fact that $\gamma_k \alpha_k \in (0, 1)$ by Theorem 7.1 and from a truncated Taylor expansion of $\log(1 - x)$, which gives $\log(1 - x) < -x$. The second equality follows from the definition of γ_t and α_t, and the final inequality follows from the fact that

$$\int_s^t k^{-(c_1+c_2)} dk \leq \sum_{k=s}^{t} k^{-(c_1+c_2)}$$

because $k^{-(c_1+c_2)}$ is a decreasing function of k.

By inspection, Equation (7.4) converges to 0 as $t \to \infty$. This in turn implies that

$$\begin{aligned} v_s(t) &= \gamma_s \gamma_t \left[\prod_{k=s+1}^{t-1} (1 - \alpha_k \gamma_k) \right] lw \\ &\leq \gamma_1^2 (st)^{-c_1} \exp\left(-\alpha_1 \gamma_1 \frac{t^{1-c_1-c_2} - s^{1-c_1-c_2}}{1 - c_1 - c_2} \right) lw \end{aligned}$$

converges to 0 as $t \to \infty$. ■

7.3.2 Constant Perturbation on Objective Functions

When the perturbation on the parameter a_i of the objective function $f_i(x_i) = a_i^{\mathrm{T}} x_i$ at time s is constant, as in the case of two adjacent cost functions parameterized by a_i and a_i', by the setup of Algorithm 7.2, the influence of this constant perturbation is bounded above by summing the influences of equivalent temporary perturbations at all iterations. Therefore, the following noise-adding mechanism masks perturbations which are constant in time and, as a result, keeps each objective function f_i differentially private. As above, since all the influences are generated by a constant perturbation, we choose to add completely correlated noise over time.

Mechanism 7.2 *In Algorithm 7.2, add completely correlated Laplacian noise*

$$v(t) = \sum_{s=1}^{t-1} v_s(t) = \begin{cases} 0, & \text{if } t = 1 \\ \gamma_1 \gamma_2 lw, & \text{if } t = 2 \\ \gamma_t \left(\gamma_{t-1} + \sum_{s=1}^{t-1} \gamma_s \prod_{k=s+1}^{t-1} (1 - \alpha_k \gamma_k) \right) lw, & \text{if } t \geq 3 \end{cases},$$

where α_t, γ_t are given in Equation (7.3)*, l is the overall Lipschitz constant for the constraint g, and $w \sim \text{Lap}(1/\varepsilon)$ is a Laplacian noise.* $\triangle$

We now present a theorem showing that Mechanism 7.2 indeed enforces differential privacy.

Theorem 7.2 *The noise-adding Mechanism 7.2 keeps the objective functions ε-differentially private in Algorithm 7.2.*

Proof: This is true by construction; see Section V in [121] where a mechanism using completely correlated noise is discussed. ■

To characterize the behavior of noise added in Algorithm 7.2, the asymptotic behavior of $v(t)$ as $t \to \infty$ is given by the following theorem.

Theorem 7.3 *In Mechanism 7.2, the noise added at time t is dominated according to*

$$v(t) \preceq \frac{\gamma_1}{\alpha_1} t^{-(c_1-c_2)} lw,$$

where constants $c_1 > c_2$ are given in Equation (7.3)*. Thus, $v(t)$ converges to 0 as $t \to \infty$.*

Proof:

By Lemma 7.1, for $t \geq 3$, expanding and neglecting terms which vanish rapidly, we find

$$\frac{v(t)}{lw} \preceq \gamma_1^2 t^{-c_1} \exp\left(-\frac{\alpha_1\gamma_1 t^{1-c_1-c_2}}{1-c_1-c_2}\right) \sum_{s=1}^{t-1} s^{-c_1} \exp\left(\frac{\alpha_1\gamma_1 s^{1-c_1-c_2}}{1-c_1-c_2}\right).$$

Noting that

$$\begin{aligned}\sum_{s=1}^{t-1} s^{-c_1} \exp\left(\frac{\alpha_1\gamma_1 s^{1-c_1-c_2}}{1-c_1-c_2}\right) &\preceq \int_1^t s^{-c_1} \exp\left(\frac{\alpha_1\gamma_1 s^{1-c_1-c_2}}{1-c_1-c_2}\right) \mathrm{d}s \\ &\preceq \frac{t^{c_2}}{\alpha_1\gamma_1} \exp\left(\frac{\alpha_1\gamma_1 t^{1-c_1-c_2}}{1-c_1-c_2}\right),\end{aligned}$$

we then find

$$v(t) \preceq \frac{\gamma_1}{\alpha_1} t^{-(c_1-c_2)} lw,$$

which converges to 0 as $t \to \infty$. ■

Having established privacy and the behavior of noise in Algorithm 7.2, the next section explores how the presence of privacy affects the behavior of the system.

7.4 Trade-Off Between Privacy and Performance

This section explores the effects of privacy upon convergence and formalizes the trade-off between these two goals. To keep objective functions differentially private, we add noise to the sequence of public multipliers $\mu^{(T)}$ according to Mechanism 7.2. This noise will, in general, prevent Algorithm 7.2 from exactly reaching an optimum, though it will still approximately reach an optimum. In this section, we show that the solution reached by Algorithm 7.2 will stay in the vicinity of the original optimum in a probabilistic sense which we state precisely below.

7.4.1 Perturbation on Public Multipliers

We first compute the influence of perturbing the public multiplier $\mu(s)$ upon the state of the i^{th} agent at time t. Let $(\mu^{x_0,\mu_0}_{D,v^{(T)}})^{(T)}$ be the sequence of public multipliers generated by Algorithm 7.2 with noise-adding mechanism $v^{(T)}$, set of objective functions D, and initial conditions x_0 and μ_0. The *s-perturbation* of $(\mu^{x_0,\mu_0}_{D,v^{(T)}})^{(T)}$ is determined by perturbing it at time s with a perturbation of size less than or equal to one, which we define now.

Definition 7.4 *An s-perturbation $\nu^{(T)}$ of $(\mu^{x_0,\mu_0}_{D,v^{(T)}})^{(T)}$ is a sequence of public multipliers such that*

- *for $k \leq s-1$, $\nu(k) = \mu^{x_0,\mu_0}_{D,v^{(T)}}(k)$,*
- *$\|\nu(s) - \mu^{x_0,\mu_0}_{D,v^{(T)}}(s)\| \leq 1$,*
- *for $k \geq s+1$, $\nu^{(T)}$ evolves by the same law as $(\mu^{x_0,\mu_0}_{D,v^{(T)}})^{(T)}$, namely by Algorithm 7.2 with noise-adding mechanism $v^{(T)}$, set of objective functions D, and initial conditions x_0 and μ_0.*

The set of s-perturbations of $(\mu^{x_0,\mu_0}_{D,v^{(T)}})^{(T)}$ is denoted by $\mathcal{Q}^s(x_0, \mu_0, D, v^{(T)})$. ◊

Using Definition 7.4, we can define the impulse response of the i^{th} agent for a perturbation on the public multiplier $\mu(s)$, given below.

Definition 7.5 *The impulse response of agent $i \in [N]$ at time $t \in [T]$ for an s-perturbation on the public multiplier is defined by*

$$\kappa_{i,s\to t} = \max_{\substack{x_0\in X,\ \mu_0\in M \\ v^{(T)}\subseteq\mathbb{R},\ \nu\in\mathcal{Q}}} |\mu_{D,0}^{x_0,\mu_0}(t) - \nu|,$$

where $\mathcal{Q} = \mathcal{Q}^s(x_0, \mu_0, D, v^{(T)})$ is the set of s-perturbations of $(\mu_{D,v^{(T)}}^{x_0,\mu_0})^{(T)}$. $\Diamond$

Definition 7.5 captures the effects of propagating forward in time a perturbation on μ. To understand the behavior of this propagation over time, Theorem 7.4, below, gives an approximate upper bound on agent i's resulting impulse response, and ensures that the upper bound converges under the following assumption.

Assumption 7.5 *The parameters α_1, γ_1 in Equation* (7.3) *satisfy*

$$\frac{\alpha_1}{\gamma_1} \gg \sum_{i=1}^{N} l_i^2,$$

where l_i is the i^{th} Lipschitz constant of the constraint g. $\Diamond$

With Assumption 7.5 in place, we now state Theorem 7.4.

Theorem 7.4 *The impulse response of agent $i \in [N]$ at time $t \in [T]$ for a perturbation on the public multiplier μ at time $s \leq t$ is bounded by*

$$\kappa_{i,s\to t} \leq \prod_{k=s+1}^{t} r_k,$$

with the rescaling factor

$$r_k = \sqrt{(1-\alpha_k\gamma_k)^2 + \gamma_k^2 \sum_{i=1}^{N} l_i^2}. \tag{7.5}$$

This upper bound is dominated according to

$$\prod_{k=s+1}^{t} r_k \preceq \exp\left(-\alpha_1\gamma_1 \frac{t^{1-c_1-c_2} - s^{1-c_1-c_2}}{1-c_1-c_2}\right),$$

which converges to 0*, as* $t \to \infty$ *under Assumption 7.5.*

Proof: Without loss of generality, set $v^{(t)} = 0$. When there is a perturbation on the public multiplier μ at time s, the influence on the state x_i of the i^{th} agent is defined as the impulse response $\kappa_{i,s\to t}$. Due to the non-expansiveness of the projection $\Pi_{X\times M}$, the propagation of $\kappa_{i,s\to t}$ is bounded by

$$\left\| \begin{bmatrix} \kappa_{i,s\to t+1} \\ \cdot \end{bmatrix} \right\| \leq \left\| \begin{bmatrix} (1-\alpha_t\gamma_t)I_N & -\gamma_t b \\ \gamma_t b^{\mathrm{T}} & 1-\alpha_t\gamma_t \end{bmatrix} \begin{bmatrix} \kappa_{i,s\to t} \\ \cdot \end{bmatrix} \right\|,$$

starting from

$$\kappa_{i,s\to s} = 1,$$
$$\tau_{i,s\to s} = [0,\ldots,0]^{\mathrm{T}}.$$

The action of

$$\begin{bmatrix} (1-\alpha_t\gamma_t)I_N & -\gamma_t b \\ \gamma_t b^{\mathrm{T}} & 1-\alpha_t\gamma_t \end{bmatrix}$$

is the composition of rotation and rescaling by a factor of

$$r_t = \sqrt{(1-\alpha_t\gamma_t)^2 + \sum_{i=1}^{N} \gamma_t^2 l_i^2}, \tag{7.6}$$

and, iterating such actions over time, we immediately arrive at the upper bound

$$\kappa_{i,s\to t} \leq \prod_{k=s+1}^{t} r_k.$$

When Assumption 7.5 is satisfied, we have

$$\frac{\alpha_t}{\gamma_t} = \frac{\alpha_1 t^{c_1 - c_2}}{\gamma_1} \gg \sum_{i=1}^{N} l_i^2, \quad t \in [T].$$

This implies that $\alpha_t \gamma_t \gg \sum_{i=1}^{N} \gamma_t^2 l_i^2$ in Equation (7.6), and consequently

$$r_t \approx 1 - \alpha_t \gamma_t.$$

In this case, using Lemma 7.1, we have

$$\prod_{k=s+1}^{t} r_k \preceq \exp\left(-\alpha_1 \gamma_1 \frac{t^{1-c_1-c_2} - s^{1-c_1-c_2}}{1 - c_1 - c_2}\right)$$

as $t \to \infty$. ■

With this result, we now quantify the trade-off between privacy and performance.

7.4.2 Trade-Off Between Privacy and Performance

To derive an upper bound on the influence of Mechanism 7.2 on the final result derived by Algorithm 7.2, it suffices to collect the influence of noise added at each iteration, which we do in the following theorem.

Theorem 7.5 *The loss of performance (see Definition 7.3) for protecting the private data D with Mechanism 7.2 is bounded by*

$$\begin{aligned} \Lambda_D &\leq \sum_{t=1}^{T-1} v(t) \kappa_{i, t \to T} \\ &\leq \gamma_1 \gamma_2 l w \prod_{k=3}^{T} r_k + \sum_{t=3}^{T-1} \gamma_t \prod_{k=t+1}^{T} r_k \left(\gamma_{t-1} + \sum_{s=1}^{t-1} \gamma_s \prod_{k=s+1}^{t-1} (1 - \alpha_k \gamma_k)\right), \end{aligned} \tag{7.7}$$

where α_t, γ_t and r_k are given in Equations (7.3) *and* (7.5)*.*

Proof: The right-hand side of Equation (7.7) follows from Definition 7.3, and the remainder is

simply expanding $v(t)$ and $\kappa_{i,t\to T}$ according to their definitions. ■

Combining Theorem 7.3 and Theorem 7.4 gives an explicit upper bound on the loss of performance Λ_D under Assumption 7.5, which we present now.

Theorem 7.6 *The loss of performance for a private data set D is bounded by*

$$\Lambda_D \leq \frac{2T^{2c_2}l}{\alpha_1^2\varepsilon^2},$$

for large T, under Assumption 7.5.

Proof: By Theorem 7.3 and Theorem 7.4 we have

$$\begin{aligned}\sum_{t=1}^{T-1} v(t)\kappa_{i,t\to T} &\preceq \sum_{t=1}^{T-1} \frac{\gamma_1 lw}{\alpha_1 t^{c_1-c_2}} \exp\left(-\alpha_1\gamma_1 \frac{T^{1-c_1-c_2}-t^{1-c_1-c_2}}{1-c_1-c_2}\right)\\ &\preceq \frac{\gamma_1 lw}{\alpha_1} \exp\left(-\frac{\alpha_1\gamma_1 T^{1-c_1-c_2}}{1-c_1-c_2}\right)\int_1^T t^{c_2-c_1}\exp\left(\frac{\alpha_1\gamma_1 t^{1-c_1-c_2}}{1-c_1-c_2}\right)\mathrm{d}t\\ &\preceq \frac{T^{2c_2}lw}{\alpha_1^2},\end{aligned}$$

under Assumption 7.5. This implies that

$$\Lambda_D \leq \frac{2T^{2c_2}l}{\alpha_1^2\varepsilon^2}$$

for large T. ■

By Theorem 7.6, the upper bound on Λ_D depends only on α_1 and not on γ_1 because γ_t decreases faster and is therefore dominated by α_t in the long run. Since this upper bound is proportional to T^{2c_2}, it is preferable to choose $c_2 \ll 1$, which is indeed compatible with Theorem 7.1. Finally, noting that the upper bound is proportional to $1/\varepsilon^2$, we see a trade-off between privacy and performance: when ε decreases, a higher degree of privacy is achieved at the cost of a larger deviation from the optimum, and *vice versa*, just as was seen in Chapter 5. For this reason, Theorem 7.6 can roughly be understood as demonstrating that a T^{2c_2} penalty is paid in convergence in exchange for a linear improvement in the privacy parameter.

7.5 Simulation Results

We now present simulation results to demonstrate the preceding bound on the loss of performance when keeping agents' objectives private. The simulation consists of $N = 6$ agents, each with state $x_i \in \mathbb{R}^2$. The values of a_i for the agents' objectives and b_i for the constraint are shown in Table 7.1. The privacy parameter was chosen to be $\varepsilon = \log 3$. In accordance with Assumption 7.5, the values $\alpha_1 = 1$ and $\gamma_1 = 0.01$ were selected, along with $c_1 = 1/2$ and $c_2 = 2/5$. Together, these give the time-varying regularization and stepsize values

$$\alpha_t = t^{-2/5} \text{ and } \gamma_t = 0.01t^{-1/2}.$$

The initial state and dual value were chosen to be $x(0) = 0$ and $\mu(0) = 0$. Using these problem parameters, two simulation runs were conducted for $T = 2,000$ iterations, one with noise and one without, in order to compare the values of $\mu_{D,v^{(T)}}^{x_0,\mu_0}(t)$ and $\mu_{D,0}^{x_0,\mu_0}(t)$ to assess performance loss.

Figure 7.2 shows the values of $\mu_{D,v^{(T)}}^{x_0,\mu_0}(t)$ (lower curve) and $\mu_{D,0}^{x_0,\mu_0}(t)$ (upper curve) across the range $1000 \leq t \leq 2000$. Here we find that, despite the noise added in accordance with Mechanism 7.2, the two trajectories are very close, with the distance between them a small fraction of their values. Figure 7.2 suggests that the level of performance loss between $\mu_{D,v^{(T)}}^{x_0,\mu_0}(t)$ and $\mu_{D,0}^{x_0,\mu_0}(t)$ is quite small, with values differing only slightly across the time horizon shown.

To further this point, Figure 7.3 shows the value of

$$\left| \frac{\mu_{D,v^{(T)}}^{x_0,\mu_0}(t) - \mu_{D,0}^{x_0,\mu_0}(t)}{\Lambda_D(t)} \right|,$$

i.e., normalized performance loss, with $\Lambda_D(t)$ computed according to Theorem 7.5 by evaluating Λ_D at each point in time during the simulation. In Figure 7.3 it can be seen that the performance loss incurred by privacy is bounded by approximately $0.05\Lambda_D(t)$ for all time, indicating close agreement between the private run and non-private run. From Figures 7.2 and 7.3, we see that privacy of the agents' objective functions is achieved in a way that still allows the optimization

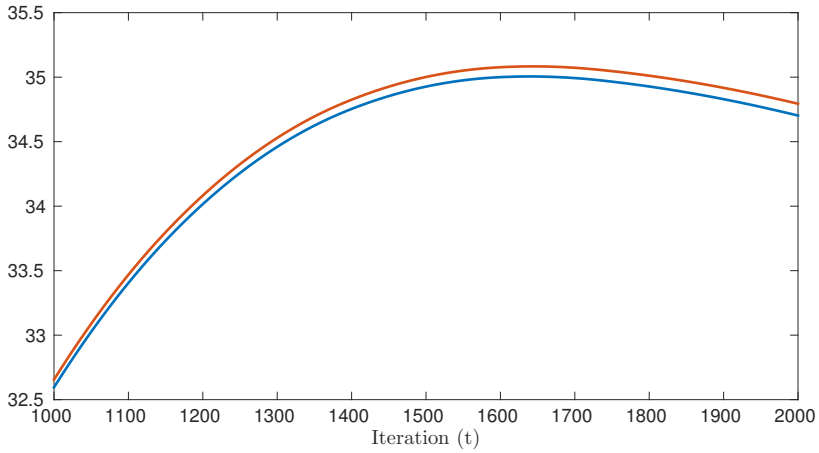

Figure 7.2: The dual trajectory $\mu_{D,v^{(T)}}^{x_0,\mu_0}(t)$ from the private optimization run (lower curve), and the dual trajectory $\mu_{D,0}^{x_0,\mu_0}(t)$ from the non-private optimization run (upper curve) for values of t between $1,000$ and $2,000$. The small difference between these two curves indicates that the results of the private optimization algorithm closely match those of the non-private algorithm. Therefore, there is only small performance loss incurred by keeping agents' objective functions private.

Table 7.1: The values of the constants a_i and b_i for $i \in [6]$, which define the objectives and constraints for simulating Algorithm 7.2.

i	a_i	b_i
1	$[-50 \quad -50]^T$	$[1.2 \quad 1.8]^T$
2	$[-30 \quad -20]^T$	$[1.5 \quad 1.3]^T$
3	$[-10 \quad -40]^T$	$[1.0 \quad 0.9]^T$
4	$[-10 \quad -60]^T$	$[1.6 \quad 1.8]^T$
5	$[-30 \quad -20]^T$	$[1.7 \quad 1.4]^T$
6	$[-100 \quad -50]^T$	$[1.8 \quad 1.3]^T$

process to proceed successfully.

7.6 Conclusion

In this work, we studied the problem of randomizing a cloud-based distributed optimization algorithm by adding noise to provide differential privacy to agents' individual objective functions. To design a noise-adding mechanism, we analyzed the impulse responses of the multi-agent system, derived an upper bound on these impulse responses, and then designed a noise-adding mechanism that keeps the objective functions differentially private in the context of a cloud-based optimiza-

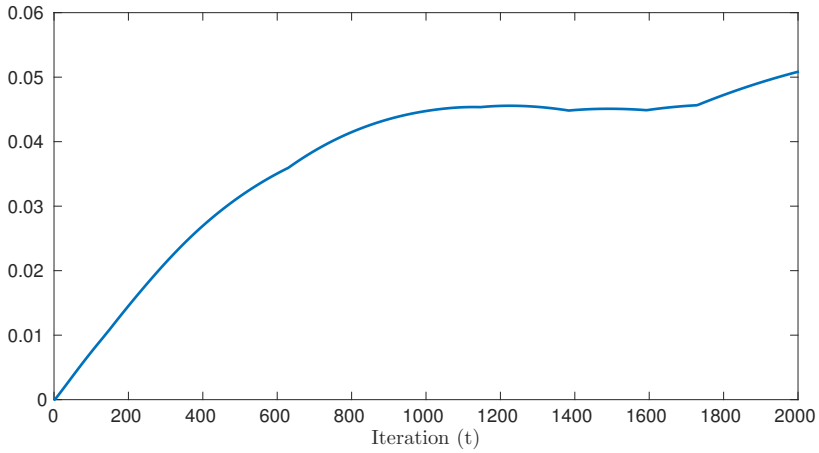

Figure 7.3: The value of $\left|\left(\mu_{D,v^{(T)}}^{x_0,\mu_0}(t) - \mu_{D,0}^{x_0,\mu_0}(t)\right)/\Lambda_D(t)\right|$ for times t between 1 and $2,000$. Here we see that the performance loss is not only bounded above by $\Lambda_D(t)$ as shown in Theorem 7.5, but that it is almost always bounded above by $0.05\Lambda_D(t)$ in this case. This small value of performance loss indicates that the private optimization algorithm is closely matching the behavior of the non-private algorithm while providing privacy guarantees to the agents' objectives.

tion algorithm. In addition, we showed that there is a trade-off between the privacy of objective functions and the performance of the randomized cloud-based optimization algorithm. Finally, the theoretical results were verified by simulation which demonstrated only minimal performance loss in exchange for enforcing user privacy.

This work took advantage of the update law used in order to add diminishing noise over time, retaining the privacy guarantees of the Laplace mechanism while minimizing the impact of privacy upon the behavior of the algorithm. In addition, by using completely correlated noise over time, this work benefits from reductions in entropy, making it possible to simultaneously protect user objective functions while permitting a network operator to perform accurate global analysis of the network.

CHAPTER 8

PRIVATE COORDINATION OF ROBOTIC NETWORKS

In this chapter we detail an experimental implementation of the work in Chapter 5. As in Chapter 4, this chapter serves both as additional testing of the work in Chapter 5 and as a demonstration of its utility in practice. Below, Section 8.1 discusses the optimization problems solved by the agents, Section 8.2 discusses the experimental platform and privacy setup used in this experiment, and Section 8.3 gives an overview of the experimental results obtained.

8.1 Optimization Problems for Terrestrial Robot Teams

There are $N = 8$ agents used in this case, and they jointly solve a sequence of three optimization problems. Each problem was crafted to have the agents assemble a particular formation. The agents are planar and their heading does not factor into these optimization problems, giving agent i a state vector $x_i \in \mathbb{R}^2$.

The first problem has the agents assemble a diagonal line in the xy-plane.

Problem 8.1

$$\begin{aligned} &\text{minimize } f(x) \\ &\text{subject to } g(x) \leq 0 \\ &x \in X, \end{aligned}$$

where

$$f(x) = \frac{1}{2}\Big(\|x_1 - t_1\|^2 + \|x_2 - t_2\|^2 + \|x_3 - t_3\|^2 + \|x_4 - t_4\|^2 + \|x_5 - t_5\|^2 + \|x_6 - t_6\|^2 + \|x_7 - t_7\|^2 + \|x_8 - t_8\|^2\Big),$$

with

$$t_1 = \begin{pmatrix} -0.55 \\ 0.30 \end{pmatrix}, t_2 = \begin{pmatrix} -0.40 \\ 0.22 \end{pmatrix}, t_3 = \begin{pmatrix} -0.25 \\ 0.14 \end{pmatrix}, t_4 = \begin{pmatrix} -0.10 \\ 0.06 \end{pmatrix},$$

$$t_5 = \begin{pmatrix} 0.05 \\ -0.02 \end{pmatrix}, t_6 = \begin{pmatrix} 0.20 \\ -0.10 \end{pmatrix}, t_7 = \begin{pmatrix} 0.35 \\ -0.18 \end{pmatrix}, t_8 = \begin{pmatrix} 0.50 \\ -0.26 \end{pmatrix},$$

$$g(x) = \begin{pmatrix} \frac{1}{10}\left(\|x_1 - x_2\|^2 - 0.05^2\right) \\ \frac{1}{10}\left(\|x_2 - x_3\|^2 - 0.05^2\right) \\ \frac{1}{10}\left(\|x_3 - x_4\|^2 - 0.05^2\right) \\ \frac{1}{10}\left(\|x_4 - x_5\|^2 - 0.05^2\right) \\ \frac{1}{10}\left(\|x_5 - x_6\|^2 - 0.05^2\right) \\ \frac{1}{10}\left(\|x_6 - x_7\|^2 - 0.05^2\right) \\ \frac{1}{10}\left(\|x_7 - x_8\|^2 - 0.05^2\right) \end{pmatrix},$$

and

$$X = \prod_{i \in [8]} [-0.6, 0.6] \times [-0.35, 0.35].$$

♦

The second problem has the agents form two squares side-by-side in the xy-plane, with each agent at the vertex of one of the squares.

Problem 8.2

$$\text{minimize } f(x)$$

$$\text{subject to } g(x) \leq 0$$

$$x \in X,$$

where

$$f(x) = \frac{1}{2}\Big(\|x_1 - t_1\|^2 + \|x_2 - t_2\|^2 + \|x_3 - t_3\|^2 + \|x_4 - t_4\|^2 \\ + \|x_5 - t_5\|^2 + \|x_6 - t_6\|^2 + \|x_7 - t_7\|^2 + \|x_8 - t_8\|^2\Big),$$

with

$$t_1 = \begin{pmatrix} -0.6 \\ 0.35 \end{pmatrix}, t_2 = \begin{pmatrix} 0 \\ 0.35 \end{pmatrix}, t_3 = \begin{pmatrix} -0.6 \\ -0.35 \end{pmatrix}, t_4 = \begin{pmatrix} 0 \\ -0.35 \end{pmatrix},$$
$$t_5 = \begin{pmatrix} 0 \\ 0.35 \end{pmatrix}, t_6 = \begin{pmatrix} 0.6 \\ 0.35 \end{pmatrix}, t_7 = \begin{pmatrix} 0 \\ -0.35 \end{pmatrix}, t_8 = \begin{pmatrix} 0.6 \\ -0.35 \end{pmatrix},$$

$$g(x) = \begin{pmatrix} \frac{1}{10}\left(\|x_1 - x_2\|^2 - 0.05^2\right) \\ \frac{1}{10}\left(\|x_2 - x_4\|^2 - 0.05^2\right) \\ \frac{1}{10}\left(\|x_3 - x_4\|^2 - 0.05^2\right) \\ \frac{1}{10}\left(\|x_1 - x_3\|^2 - 0.05^2\right) \\ \frac{1}{10}\left(\|x_5 - x_6\|^2 - 0.05^2\right) \\ \frac{1}{10}\left(\|x_6 - x_8\|^2 - 0.05^2\right) \\ \frac{1}{10}\left(\|x_5 - x_7\|^2 - 0.05^2\right) \\ \frac{1}{10}\left(\|x_7 - x_8\|^2 - 0.05^2\right) \end{pmatrix},$$

and

$$X = \prod_{i \in [8]} [-0.6, 0.6] \times [-0.35, 0.35].$$

♦

The third problem has the agents form a diamond centered at the origin in the xy-plane.

Problem 8.3

$$\text{minimize } f(x)$$

$$x \in X,$$

where

$$f(x) = \frac{1}{2}\Big(\|x_1 - t_1\|^2 + \|x_2 - t_2\|^2 + \|x_3 - t_3\|^2 + \|x_4 - t_4\|^2 + \|x_5 - t_5\|^2 + \|x_6 - t_6\|^2 + \|x_7 - t_7\|^2 + \|x_8 - t_8\|^2\Big),$$

with

$$t_1 = \begin{pmatrix} 0 \\ 0.2 \end{pmatrix}, t_2 = \begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix}, t_3 = \begin{pmatrix} 0.2 \\ 0 \end{pmatrix}, t_4 = \begin{pmatrix} 0.1 \\ -0.1 \end{pmatrix},$$

$$t_5 = \begin{pmatrix} 0 \\ -0.1 \end{pmatrix}, t_6 = \begin{pmatrix} -0.1 \\ -0.1 \end{pmatrix}, t_7 = \begin{pmatrix} -0.1 \\ 0 \end{pmatrix}, t_8 = \begin{pmatrix} -0.1 \\ 0.1 \end{pmatrix},$$

and

$$X = \prod_{i \in [8]} [-0.6, 0.6] \times [-0.35, 0.35].$$

♦

8.2 Experimental Platform and Setup

The experiments in this chapter were run on the Robotarium [15], a testbed for swarm robotics research[1]. The Robotarium features an architecture similar to that used in Chapter 4, namely a central base station which sends position commands to each robot, rather than point-to-point communications among the agents. In this case, position information comes from an overhead camera

[1] http://www.robotarium.org

which identifies agents based on unique fiducials placed on top of each robot. The Robotarium has a rectangular workspace for the robots, with the origin at the center of the rectangle and its corners at the coordinates $(-0.60, -0.35)$, $(-0.60, 0.35)$, $(0.60, -0.35)$, and $(0.60, 0.35)$, giving rise to the choice of constraint set X above in Problems 8.1, 8.2, and 8.3.

The experiments in this chapter were run using eight GRITSBots [122], each of which is a two-wheeled differential drive robot roughly 3cm across with Dubins vehicle [123] dynamics, i.e., its position (x, y) and heading θ evolve according to

$$\begin{aligned}\dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \omega,\end{aligned}$$

where the pair (v, ω) is the control input to each robot. Algorithm 5.2 generates positions rather than pairs (v, ω). To translate desired positions into linear and angular velocity commands, one can implement a proportional controller on a virtual point some fixed distance ℓ in front of the robot and then have the robot track that point. Here $\ell = 5$cm was chosen, and this transformation was computed using built-in functions provided by the Robotarium.

In this implementation, ε-differential privacy was used, and the privacy parameter was set to $\varepsilon = \log 2$, which is within the range of privacy parameters typically used in the literature. The adjacency parameter was set to $B = 0.2$. Given the dimensions of the Robotarium, this choice of B is reasonable as it specifies that adjacent trajectories may deviate from each other by up to 20cm over time. As in Chapter 4, control barrier functions [74] were used for collision avoidance. The safety radius was set to 6cm, ensuring that all pairs of robots had their centers of mass at least 6cm apart at all times.

8.3 Experimental Results

The total time taken to solve Problems 8.1, 8.2, and 8.3 was 40.9 seconds, and the time required to solve each individual problem is shown in Table 8.1.

Table 8.1: The experimental time required for eight GRITSBots to privately solve Problems 8.1, 8.2, and 8.3 using Algorithm 5.2.

Task	Runtime (s)
Solve Problem 8.1	14.5
Solve Problem 8.2	13.0
Solve Problem 8.3	13.4

The initial position of the GRITSBots was a random configuration in the workspace, shown from above in Figure 8.1.

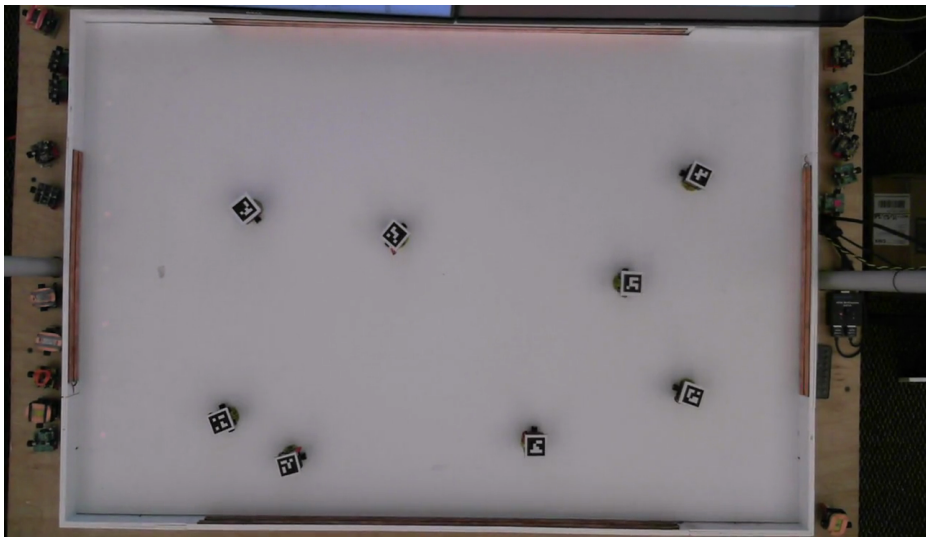

Figure 8.1: A top-down view of the initial position of eight GRITSBots when they begin using Algorithm 5.2.

After 14.5 seconds, the GRITSBots have arrived at their solution to Problem 8.1, shown in Figure 8.2. Here we see that the robots do indeed assemble the desired formation despite the noise added for privacy.

Figure 8.2: A top-down view of eight GRITSBots at their collective solution to Problem 8.1. As desired, they form a diagonal line across their workspace.

Upon solving Problem 8.1, the GRITSBots immediately switch to solving Problem 8.2, which takes 13.0s. The resulting formation is shown in Figure 8.3 where the robots have arrived at the expected formation of two side-by-side squares.

Figure 8.3: A top-down view of eight GRITSBots at their collective solution to Problem 8.2. As desired, they assemble two adjacent square formations.

When Problem 8.2 is solved, the GRITSBots then solve Problem 8.3. Their collective solution to Problem 8.3 is shown in Figure 8.4, where they can be seen to have formed the desired diamond shape at the origin.

Figure 8.4: A top-down view of eight GRITSBots at their collective solution to Problem 8.3. As desired, they form a single diamond-shaped formation centered at the origin.

8.4 Discussion and Conclusions

The robotic experiments in this chapter demonstrate rapid and accurate convergence of Algorithm 5.2 in this implementation. None of the runtimes for solving Problems 8.1, 8.2, or 8.3 are particularly long even though each formation is quite different from the others. These results indicate that trajectory-level privacy can be integrated into practical systems without substantially harming their performance.

These experiments also indicate that Algorithm 5.2 is particularly well-suited to practical applications in several ways. First, Algorithm 5.2 generates only position information, and these positions are first mapped to a proportional controller for a point mass, and the resulting controller for that system is then mapped to the non-holonomic unicycle dynamics of the GRITSBots. In spite of these transformations, the problems in this chapter are all solved successfully. Second,

the barrier functions used for collision avoidance in this chapter introduce errors into each robot's position by preventing it from arriving exactly at its desired position. Nonetheless, the agents successfully solve each problem, indicating that this work is well-suited to practical applications, even when there are obstacles, position errors, and other factors that can affect an agent's performance.

Part III

Conclusions

CHAPTER 9

CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

This thesis presented algorithms that take advantage of a mixed centralized/decentralized architecture to coordinate networks of agents under challenging conditions. This architecture is enabled by recent technological developments which allow for both the large-scale production of many low-cost devices to coordinate, and the seamless inclusion of centralized cloud computing into many systems. Given the resulting technological feasibility of a mixed centralized/decentralized system, this work was focused on exploiting this architecture and blending the conventionally-distinct paradigms of centralized and decentralized coordination. This was done successfully for two classes of problems that arise naturally in networks, namely asynchronous coordination and coordination under privacy requirements in Parts I and II, respectively.

Part I demonstrated successful mixed centralized/decentralized coordination of teams of agents even in the presence of total disagreement about the state of the network. While the degree of asynchrony allowed in the system inevitably leads to the accumulation of certain errors in the system over time, the choice of gradient update law in the asynchronous coordination framework successfully mitigates these errors, reducing them to small enough levels that the error in the system is tolerable in many applications of interest. The asynchronous framework in Chapter 2 is general enough to accommodate essentially any communications structure in the system, making it well-suited to practical applications, including those in which communications are intermittent and unreliable. Chapter 3 analyzed random communications in the setting of this asynchronous framework, letting a network operator predict the behavior of the system in terms of network parameters. And this work was observed to work well in practice in Chapter 4. There, it was observed that teams of quadrotors can indeed make use of this work and successfully assemble formations even when their communications and computations are randomized.

Part II demonstrated mixed centralized/decentralized coordination of teams of agents under several forms of privacy. There is a natural tension between privacy, which seeks to conceal information, and network coordination, which seeks to share information. It was shown that, in several ways, noise can be added to afford individual users strong privacy guarantees and that, although the magnitude of noise added can be much larger than other terms in the problems considered, the choice of gradient-based update law helps create algorithms that are robust to this noise. This was demonstrated in Chapter 5 where individual agents' entire state trajectories were kept private, yet the agents were still able to work together and attain useful solutions to a broad class of optimization problems. This work was shown in Chapter 6 to also promote truthful information reporting by the agents because it largely removes incentives they have to share false information with the cloud. Chapter 7 then provided an algorithm to keep agents' objective functions private as they work together, and all three chapters found that linear changes in the network's privacy parameter result in an inverse square penalty in convergence. This work was successfully demonstrated on ground robots in Chapter 8, where it was seen that a team of eight wheeled robots can successfully work together, even when they are keeping their exact trajectories private.

One possible future direction for this work is to develop techniques for providing individual agents with privacy guarantees while still allowing for asynchronous communications. This is of interest not only because it unifies two problem domains of interest to the network control community, but also because it can help understand the boundary between what is possible with networks and what is not. The challenge of private asynchronous coordination is, naturally, that information can be both severely delayed and corrupted by noise, leading to even greater disagreements about a network's state than were seen in this work. It is not clear exactly what form a private asynchronous coordination framework may take, though the work in this thesis represents steps along the way to such a framework and, ultimately, a comprehensive understanding of the capabilities of networked systems.

Another direction for future work concerns the deployment of this work to physical systems beyond robots. Certainly there are many more applications in robotics than can be explored, but

beyond that, several emerging applications could make use of this work. These applications include smart and connected communities, smart power grids, and autonomous vehicles. In all three cases, real-time safety guarantees become vital to system performance, and preventing constraint violation at all iterates of an optimization algorithm becomes essential. It is not yet clear how real-time constraint checking can be incorporated into these systems, though these emerging applications (and others) will continue to be a rich source of network coordination problems for the foreseeable future.

REFERENCES

[1] M. Khan, G. Pandurangan, and V. Kumar, "Distributed algorithms for constructing approximate minimum spanning trees in wireless sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 1, pp. 124–139, 2009.

[2] N. Trigoni and B. Krishnamachari, "Sensor network algorithms and applications: Introduction," *Philosophical Transactions of the Royal Society A - Mathematical, Physical, and Engineering Sciences*, vol. 370, no. 1958, pp. 5–10, 2012.

[3] J. Cortés, S. Martinez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *Proceedings of the 2002 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, 2002, pp. 1327–1332.

[4] M. Rabbat and R. Nowak, "Distributed optimization in sensor networks," *Proceedings of the Third International Symposium on Information Processing in Sensor Networks (IPSN)*, 2004, pp. 20–27.

[5] Y. Guo and L. E. Parker, "A distributed and optimal motion planning approach for multiple mobile robots," *Proceedings of the 2002 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3, 2002, pp. 2612–2619.

[6] D. E. Soltero, M. Schwager, and D. Rus, "Decentralized path planning for coverage tasks using gradient descent adaptive control," *The International Journal of Robotics Research*, vol. 33, no. 3, pp. 401–425, 2014.

[7] P. Vytelingum, T. D. Voice, S. D. Ramchurn, A. Rogers, and N. R. Jennings, "Agent-based micro-storage management for the smart grid," *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2010, pp. 39–46.

[8] S. Caron and G. Kesidis, "Incentive-based energy consumption scheduling algorithms for the smart grid," *Proceedings of the First IEEE International Conference on Smart Grid Communications (SmartGridComm)*, 2010, pp. 391–396.

[9] F. Kelly, A. Maulloo, and D. Tan, "Rate control in communication networks: Shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, no. 3, pp. 237–252, 1998.

[10] M. Chiang, S. Low, A. Calderbank, and J. Doyle, "Layering as optimization decomposition: A mathematical theory of network architectures," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 255–312, 2007.

[11] D. Mitra, "An asynchronous distributed algorithm for power control in cellular radio systems," in *Wireless and Mobile Communications*, J. M. Holtzman and D. J. Goodman, Eds. Boston, MA: Springer, 1994, pp. 177–186.

[12] R. Baheti and H. Gill, "Cyber-physical systems," in *The Impact of Control Technology*, T. Samad and A. Annaswamy, Eds. New York, NY: IEEE Control Systems Society, 2011, vol. 12, pp. 161–166.

[13] J. N. Tsitsiklis, "Problems in decentralized decision making and computation," PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1984.

[14] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The GRASP multiple micro-UAV testbed," *IEEE Robotics and Automation Magazine*, vol. 17, no. 3, pp. 56–65, 2010.

[15] D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. Ames, E. Feron, and M. Egerstedt, "The Robotarium: A remotely accessible swarm robotics research testbed," *Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA)*, To appear, 2017.

[16] United States Department of Energy, Office of the General Counsel, "Data access and privacy issues related to smart grid technologies," US Department of Energy, Washington, DC, Tech. Rep., 2010.

[17] The European Data Protection Supervisor, "Opinion of the European Data Protection Supervisor on the commission recommendation on preparations for the roll-out of smart metering systems," European Union, Brussels, Belgium, Tech. Rep., 2012.

[18] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Belmont, MA: Athena Scientific, 1997.

[19] D. P. Bertsekas and J. N. Tsitsiklis, "Convergence rate and termination of asynchronous iterative algorithms," *Proceedings of the 3rd International Conference on Supercomputing (ICS)*, 1989, pp. 461–470.

[20] D. P. Bertsekas, "Distributed asynchronous computation of fixed points," *Mathematical Programming*, vol. 27, no. 1, pp. 107–120, 1983.

[21] D. Chazan and W. Miranker, "Chaotic relaxation," *Linear Algebra and its Applications*, vol. 2, no. 2, pp. 199 –222, 1969.

[22] G. M. Baudet, "Asynchronous iterative methods for multiprocessors," *Journal of the ACM*, vol. 25, no. 2, pp. 226–244, 1978.

[23] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Transactions on Automatic Control*, vol. 49, no. 9, pp. 1520–1533, 2004.

[24] M. Zhu and S. Martinez, "On distributed convex optimization under inequality and equality constraints," *IEEE Transactions on Automatic Control*, vol. 57, no. 1, pp. 151–164, 2012.

[25] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE/ACM Transactions on Networking*, vol. 14, pp. 2508–2530, Jun. 2006.

[26] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, Jan. 2011.

[27] R. Zhang and J. T. Kwok, "Asynchronous distributed ADMM for consensus optimization," *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 2014, pp. 1701–1709.

[28] D. P. Bertsekas, A. Nedić, and A. E. Ozdaglar, *Convex analysis and optimization*, ser. Athena Scientific Optimization and Computation series. Belmont, MA: Athena Scientific, 2003.

[29] H. W. Kuhn and A. W. Tucker, "Nonlinear programming," *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, 1951, pp. 481–492.

[30] F. Facchinei and J.-S. Pang, *Finite-dimensional variational inequalities and complementarity problems*. New York, NY: Springer Verlag, 2003, vol. 1 and 2.

[31] H. Uzawa, "Iterative methods in concave programming," in *Studies in Linear and Non-Linear Programming*, K. Arrow, L. Hurwicz, and H. Uzawa, Eds. Stanford, CA: Stanford University Press, 1958, pp. 154–165.

[32] M. T. Hale, A. Nedić, and M. Egerstedt, "Cloud-based centralized/decentralized multi-agent optimization with communication delays," *Proceedings of the 54th IEEE Conference on Decision and Control (CDC)*, 2015, pp. 700–705.

[33] J. Koshal, A. Nedić, and U. V. Shanbhag, "Multiuser optimization: Distributed algorithms and error analysis," *SIAM Journal on Optimization*, vol. 21, no. 3, pp. 1046–1081, 2011.

[34] A. Bakushinskii and B. Polyak, "Solution of variational inequalities," *Doklady Akademii Nauk SSSR*, vol. 219, no. 5, pp. 1038–1041, 1974.

[35] D. E. Comer, *Internetworking with TCP/IP, Volume 1: Principles, Protocols, and Architectures*, Fourth Edition. Upper Saddle River, NJ: Prentice Hall, 2000.

[36] B. T. Polyak, *Introduction to optimization*. New York, NY: Optimization Software, Inc., 1987.

[37] M. Mesbahi and M. Egerstedt, *Graph theoretic methods in multiagent networks*. Princeton, NJ: Princeton University Press, 2010.

[38] P. Erdős and A. Rényi, "On random graphs I," *Publicationes Mathematicae Debrecen*, vol. 6, pp. 290–297, 1959.

[39] Y. Hatano and M. Mesbahi, "Agreement over random networks," *IEEE Transactions on Automatic Control*, vol. 50, no. 11, pp. 1867–1872, 2005.

[40] M. Fiedler, "Algebraic connectivity of graphs," *Czechoslovak Mathematical Journal*, vol. 23, no. 2, pp. 298–305, 1973.

[41] L. Moreau, "Stability of multiagent systems with time-dependent communication links," *IEEE Transactions on Automatic Control*, vol. 50, no. 2, pp. 169–182, 2005.

[42] W. Ren and R. W. Beard, "Consensus seeking in multiagent systems under dynamically changing interaction topologies," *IEEE Transactions on Automatic Control*, vol. 50, no. 5, pp. 655–661, 2005.

[43] P. Tseng, D. P. Bertsekas, and J. N. Tsitsiklis, "Partially asynchronous, parallel algorithms for network flow and other problems," *SIAM Journal on Control and Optimization*, vol. 28, no. 3, pp. 678–710, 1990.

[44] P. Tseng, "On the rate of convergence of a partially asynchronous gradient projection algorithm," *SIAM Journal on Optimization*, vol. 1, no. 4, pp. 603–619, 1991.

[45] W. Ren, R. W. Beard, and T. W. McLain, "Coordination variables and consensus building in multiple vehicle systems," in *Cooperative Control: A Post-Workshop Volume, 2003 Block Island Workshop on Cooperative Control*, V. Kumar, N. Leonard, and A. S. Morse, Eds., ser. Lecture Notes in Control and Information Sciences. Berlin, GER: Springer Science+Business Media, 2004, vol. 309.

[46] A. Nedić, A. Ozdaglar, and P. A. Parrilo, "Constrained consensus and optimization in multi-agent networks," *IEEE Transactions on Automatic Control*, vol. 55, no. 4, pp. 922–938, 2010.

[47] A. Nedić and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

[48] A. Olshevsky and J. N. Tsitsiklis, "Convergence speed in distributed consensus and averaging," *SIAM Review*, vol. 53, no. 4, pp. 747–772, 2011.

[49] H. R. Feyzmahdavian and M. Johansson, "On the convergence rates of asynchronous iterations," *Proceedings of the 53rd IEEE Conference on Decision and Control (CDC)*, 2014, pp. 153–159.

[50] A. Nedić and A. Ozdaglar, "On the rate of convergence of distributed subgradient methods for multi-agent optimization," *Proceedings of the 46th IEEE Conference on Decision and Control (CDC)*, 2007, pp. 4711–4716.

[51] A. I. Chen and A. Ozdaglar, "A fast distributed proximal-gradient method," *Proceedings of the 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2012, pp. 601–608.

[52] V. Blondel, J. M. Hendrickx, A. Olshevsky, and J. Tsitsiklis, "Convergence in multiagent coordination, consensus, and flocking," *Proceedings of the 44th IEEE Conference on Decision and Control (CDC)*, 2005, pp. 2996–3000.

[53] A. Jadbabaie, J. Lin, and A. S. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Transactions on Automatic Control*, vol. 48, no. 6, pp. 988–1001, 2003.

[54] B. Touri and A. Nedić, "Distributed consensus over network with noisy links," *Proceedings of the 12th International Conference on Information Fusion (FUSION)*, 2009, pp. 146–154.

[55] S. S. Kia, J. Cortés, and S. Martinez, "Distributed event-triggered communication for dynamic average consensus in networked systems," *Automatica*, vol. 59, no. C, pp. 112–119, Sep. 2015.

[56] P. Diaconis and M. Shahshahani, "On the eigenvalues of random matrices," *Journal of Applied Probability*, vol. 31, pp. 49–62, 1994.

[57] T. Tao, *Topics in random matrix theory*, ser. Graduate Studies in Mathematics. Providence, RI: American Mathematical Society, 2012, vol. 132.

[58] N. Alon, M. Krivelevich, and V. H. Vu, "On the concentration of eigenvalues of random symmetric matrices," *Israel Journal of Mathematics*, vol. 131, no. 1, pp. 259–267, 2002.

[59] Z. Füredi and J. Komlós, "The eigenvalues of random symmetric matrices," *Combinatorica*, vol. 1, no. 3, pp. 233–241, 1981.

[60] E. P. Wigner, "On the distribution of the roots of certain symmetric matrices," *Annals of Mathematics*, vol. 67, no. 2, pp. 325–327, 1958.

[61] A. Coja-Oghlan, "On the Laplacian eigenvalues of G(n, p)," *Combinatorics, Probability and Computing*, vol. 16, no. 6, pp. 923–946, 2007.

[62] F. Juhász, "On the asymptotic behaviour of the spectra of non-symmetric random (0, 1) matrices," *Discrete Mathematics*, vol. 41, no. 2, pp. 161–165, 1982.

[63] B. Bollobás, *Random Graphs*, 2nd ed., ser. Cambridge Studies in Advanced Mathematics. Cambridge, UK: Cambridge University Press, 2001, vol. 73.

[64] M. Mehyar, D. Spanos, J. Pongsajapan, S. H. Low, and R. M. Murray, "Asynchronous distributed averaging on communication networks," *IEEE/ACM Transactions on Networking*, vol. 15, no. 3, pp. 512–520, 2007.

[65] M. H. Nazari, Z. Costello, M. J. Feizollahi, S. Grijalva, and M. Egerstedt, "Distributed frequency control of prosumer-based electric energy systems," *IEEE Transactions on Power Systems*, vol. 29, no. 6, pp. 2934–2942, 2014.

[66] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh, "Deploying a wireless sensor network on an active volcano," *IEEE Internet Computing*, vol. 10, no. 2, pp. 18–25, 2006.

[67] M. Krivelevich and B. Sudakov, "The largest eigenvalue of sparse random graphs," *Combinatorics, Probability and Computing*, vol. 12, no. 1, pp. 61–72, 2003.

[68] L. V. Tran, V. H. Vu, and K. Wang, "Sparse random graphs: Eigenvalues and eigenvectors," *Random Structures & Algorithms*, vol. 42, no. 1, pp. 110–134, 2013.

[69] C. Godsil and G. Royle, *Algebraic graph theory*, ser. Graduate Texts in Mathematics. New York, NY: Springer-Verlag, 2001, vol. 207.

[70] B. C. Arnold and R. A. Groeneveld, "Bounds on expectations of linear systematic statistics based on dependent samples," *Annals of Statistics*, vol. 7, no. 1, pp. 220–223, Jan. 1979.

[71] R. E. A. C. Paley and A. Zygmund, "A note on analytic functions in the unit circle," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 28, no. 3, pp. 266–272, 1932.

[72] G. M. Hoffmann, H. Huang, S. L. Waslander, and C. J. Tomlin, "Quadrotor helicopter flight dynamics and control: Theory and experiment," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Paper number 2007-6461, 2007.

[73] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," *Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 2520–2525.

[74] L. Wang, A. D. Ames, and M. Egerstedt, "Safe certificate-based maneuvers for teams of quadrotors using differential flatness," *Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA)*, To appear, 2017.

[75] A. Nedić and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

[76] I. Lobel and A. Ozdaglar, "Distributed subgradient methods for convex optimization over random networks," *IEEE Transactions on Automatic Control*, vol. 56, no. 6, pp. 1291–1306, 2011.

[77] E. Wei, A. Ozdaglar, and A. Jadbabaie, "A distributed Newton method for network utility maximization I: Algorithm," *IEEE Transactions on Automatic Control*, vol. 58, no. 9, pp. 2162–2175, 2013.

[78] C. Dwork, "Differential privacy," *Proceedings of the 33rd International Conference on Automata, Languages and Programming (ICALP)*, 2006, pp. 1–12.

[79] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," *Proceedings of the Third Theory of Cryptography Conference (TCC)*, 2006, pp. 265–284.

[80] C. Dwork, "Differential privacy: A survey of results," *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation (TAMC)*, 2008, pp. 1–19.

[81] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3-4, pp. 211–407, 2014.

[82] J. Le Ny and G. Pappas, "Differentially private filtering," *IEEE Transactions on Automatic Control*, vol. 59, no. 2, pp. 341–354, 2014.

[83] S. P. Kasiviswanathan and A. Smith, "A note on differential privacy: Defining resistance to arbitrary side information," *arXiv Preprint. Available at: http://arxiv.org/abs/0803.3946v1*, 2008.

[84] J. Hsu, A. Roth, T. Roughgarden, and J. Ullman, "Privately solving linear programs," *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP)*, 2014, pp. 612–624.

[85] S. Han, U. Topcu, and G. Pappas, "Differentially private convex optimization with piecewise affine objectives," *Proceedings of the 53rd IEEE Conference on Decision and Control (CDC)*, 2014, pp. 2160–2166.

[86] Z. Huang, S. Mitra, and N. Vaidya, "Differentially private distributed optimization," *Proceedings of the 2015 International Conference on Distributed Computing and Networking (ICDCN)*, 2015, 4:1–4:10.

[87] Z. Huang, S. Mitra, and G. Dullerud, "Differentially private iterative synchronous consensus," *Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society (WPES)*, 2012, pp. 81–90.

[88] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: State-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[89] R. T. Rockafellar, "On the maximal monotonicity of subdifferential mappings," *Pacific Journal of Mathematics*, vol. 33, no. 1, pp. 209–216, 1970.

[90] I. Konnov, *Equilibrium models and variational inequalities*, ser. Mathematics in Science and Engineering. Amsterdam, NL: Elsevier, 2007, vol. 210.

[91] B. Poljak, "Nonlinear programming methods in the presence of noise," *Mathematical programming*, vol. 14, no. 1, pp. 87–97, 1978.

[92] A. A. Goldstein, "Convex programming in Hilbert space," *Bulletin of the American Mathematical Society*, vol. 70, no. 5, pp. 709–710, Sep. 1964.

[93] E. Levitin and B. Polyak, "Constrained minimization methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 6, no. 5, pp. 1–50, 1966.

[94] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, "Our data, ourselves: Privacy via distributed noise generation," *Proceedings of the 25th Annual International Cryptology Conference (EUROCRYPT)*, 2006, pp. 486–503.

[95] F. Yousefian, A. Nedić, and U. Shanbhag, "A regularized smoothing stochastic approximation (RSSA) algorithm for stochastic variational inequality problems," *Proceedings of the 2013 Winter Simulation Conference (WSC)*, 2013, pp. 933–944.

[96] J. Koshal, A. Nedić, and U. Shanbhag, "Regularized iterative stochastic approximation methods for stochastic variational inequality problems," *IEEE Transactions on Automatic Control*, vol. 58, no. 3, pp. 594–609, 2013.

[97] H. M. Edwards, *Riemann's zeta function*, ser. Pure and Applied Mathematics. New York, NY: Academic Press, 1974, vol. 58.

[98] D. H. Greene and D. E. Knuth, *Mathematics for the analysis of algorithms*, ser. Progress in Computer Science and Applied Logic. Boston: Birkhauser, 1981.

[99] G. Zoutendijk, "Nonlinear programming, computational methods," *Integer and Nonlinear Programming*, J. Abadie, Ed., Amsterdam, NL: North-Holland, 1970, pp. 37–86.

[100] M. Kearns, M. Pai, A. Roth, and J. Ullman, "Mechanism design in large games: Incentives and privacy," *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science (ITCS)*, 2014, pp. 403–410.

[101] S. Han, U. Topcu, and G. Pappas, "An approximately truthful mechanism for electric vehicle charging via joint differential privacy," *Proceedings of the 2015 American Control Conference (ACC)*, 2015, pp. 2469–2475.

[102] J. Hsu, Z. Huang, A. Roth, and Z. S. Wu, "Jointly private convex programming," *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2016, pp. 580–599.

[103] S. Han, U. Topcu, and G. J. Pappas, "Differentially private distributed constrained optimization," *IEEE Transactions on Automatic Control*, vol. 62, no. 1, pp. 50–64, 2017.

[104] F. McSherry and K. Talwar, "Mechanism design via differential privacy," *Proceedings of the 48th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2007, pp. 94–103.

[105] S. Braynov and M. Jadliwala, "Detecting malicious groups of agents," *Proceedings of the First IEEE Symposium on Multi-Agent Security and Survivability (MAS&S)*, 2004, pp. 90–99.

[106] A. Fagiolini, M. Pellinacci, G. Valenti, G. Dini, and A. Bicchi, "Consensus-based distributed intrusion detection for multi-robot systems," *Proceedings of the 2008 IEEE International Conference on Robotics and Automation (ICRA)*, 2008, pp. 120–127.

[107] M. T. Hale and M. Egerstedt, "Cloud-enabled differentially private multi-agent optimization with constraints," *IEEE Transactions on Control of Network Systems*, 2017, Conditionally accepted. Available at http://arxiv.org/abs/1507.04371.

[108] M. Pipattanasomporn, H. Feroze, and S. Rahman, "Multi-agent systems in a distributed smart grid: Design and implementation," *Proceedings of the 2009 IEEE/PES Power Systems Conference and Exposition (PSCE)*, 2009, pp. 1–8.

[109] P. Xuan, V. Lesser, and S. Zilberstein, "Communication decisions in multi-agent cooperation: Model and experiments," *Proceedings of the Fifth International Conference on Autonomous Agents (AGENTS)*, 2001, pp. 616–623.

[110] R. Becker, A. Carlin, V. Lesser, and S. Zilberstein, "Analyzing myopic approaches for multi-agent communication," *Computational Intelligence*, vol. 25, no. 1, pp. 31–50, 2009.

[111] S. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," *IEEE Acoustics, Speech, and Signal Processing (ASSP) Magazine*, vol. 6, no. 3, pp. 4–19, 1989.

[112] A. Dimakis, S. Kar, J. Moura, M. Rabbat, and A. Scaglione, "Gossip algorithms for distributed signal processing," *Proceedings of the IEEE*, vol. 98, no. 11, pp. 1847–1864, 2010.

[113] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous Agents and Multi-Agent Systems*, vol. 11, no. 3, pp. 387–434, 2005.

[114] Z. Huang, Y. Wang, S. Mitra, and G. E. Dullerud, "On the cost of differential privacy in distributed control systems," *Proceedings of the 3rd International Conference on High Confidence Networked Systems (HiCoNS)*, 2014, pp. 105–114.

[115] N. Mohammed, R. Chen, B. C. Fung, and P. S. Yu, "Differentially private data release for data mining," *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2011, pp. 493–501.

[116] H. Brenner and K. Nissim, "Impossibility of differentially private universally optimal mechanisms," *SIAM Journal on Computing*, vol. 43, no. 5, pp. 1513–1540, 2014.

[117] E. Nozari, P. Tallapragada, and J. Cortés, "Differentially private distributed convex optimization via objective perturbation," *Proceedings of the 2016 American Control Conference (ACC)*, 2016, pp. 2061–2066.

[118] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate, "Differentially private empirical risk minimization," *Journal of Machine Learning Research*, vol. 12, pp. 1069–1109, 2011.

[119] R. Bassily, A. Smith, and A. Thakurta, "Private empirical risk minimization: Efficient algorithms and tight error bounds," *Proceedings of the 55th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2014, pp. 464–473.

[120] M. T. Hale and M. Egerstedt, "Differentially private cloud-based multi-agent optimization with constraints," *Proceedings of the 2015 American Control Conference (ACC)*, 2015, pp. 1235–1240.

[121] Y. Wang, Z. Huang, S. Mitra, and G. E. Dullerud, "Entropy-minimizing mechanism for differential privacy of discrete-time linear feedback systems," *Proceedings of the 53rd IEEE Conference on Decision and Control (CDC)*, 2014, pp. 2130–2135.

[122] D. Pickem, M. Lee, and M. Egerstedt, "The GRITSBot in its natural habitat - A multi-robot testbed," *Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 4062–4067.

[123] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957.